IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

CISCO SYSTEMS, INC., and
CISCO TECHNOLOGY, INC.,

Plaintiffs,

v.

TELCORDIA TECHNOLOGIES, INC.

Defendant.

C. A. No. 07-113 (GMS)

JOINT APPENDIX OF INTRINSIC AND EXTRINSIC EVIDENCE

MORRIS, NICHOLS, ARSHT & TUNNELL LLP
Jack B. Blumenfeld (#1014)
Karen Jacobs Louden (#2881)
1201 North Market Street
Wilmington, DE 19899-1347
(302) 658-9200
jblumenfeld@mnat.com
klouden@mnat.com
Attorneys for Plaintiffs Cisco Systems, Inc. and Cisco Technology Inc.

OF COUNSEL:

Matthew D. Powers
Edward R. Reines
Sonal N. Mehta
WEIL, GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
(650) 802-3000

David Greenbaum
WEIL, GOTSHAL & MANGES LLP
767 Fifth Avenue
New York, NY 10153
(212) 310-8000

ASHBY & GEDDES
Steven J. Balick (#2114)
John G. Day (#2403)
Tiffany Geyer Lydon (#3950)
500 Delaware Ave., 18th Floor
P.O. Box 1150
Wilmington, DE 19899
(302) 654-1888
sbalick@ashby-geddes.com
jday@ashby-geddes.com
tlydon@ashby-gedees.com
Attorneys for Defendant Telcordia Technologies, Inc.

OF COUNSEL:

Vincent P. Kovalick
Christopher T. Blanford
FINNEGAN, HENDERSON, FARABOW, GARRETT & DUNNER, L.L.P.
901 New York Avenue, N.W.
Washington, D.C. 20001
(202) 408-4000

May 12, 2008

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

CISCO SYSTEMS, INC., and CISCO TECHNOLOGY, INC., Plaintiffs,	)	
v.	)	C. A. No. 07-113 (GMS)
TELCORDIA TECHNOLOGIES, INC. Defendant.	)	

JOINT APPENDIX OF INTRINSIC AND EXTRINSIC EVIDENCE

Tab	Description	Citing Part(ies)	Pages
1	U.S. Patent No. 5,142,622	Cisco, Telcordia	A1-A15
2	March 29, 1990 Office Action	Cisco, Telcordia	A16-A22
3	June 29, 1990 Amendment	Cisco, Telcordia	A23-A35
4	February 14, 1992 Amendment	Telcordia	A36-A49
5	January 31, 1989 Application	Telcordia	A50-A85

CERTIFICATE OF SERVICE

I certify that on May 12, 2008 I electronically filed the foregoing with the Clerk of the Court using CM/ECF, which will send notification of such filing(s) to the following:

Steven J. Balick
John G. Day
Tiffany Geyer Lydon
ASHBY & GEDDES

I further certify that I caused copies to be served upon the following on May 12, 2008 in the manner indicated:

BY HAND DELIVERY and ELECTRONIC MAIL

Steven J. Balick
John G. Day
Tiffany Geyer Lydon
ASHBY & GEDDES
500 Delaware Avenue, 8th Floor
Wilmington, DE 19801

BY ELECTRONIC MAIL

Vincent P. Kovalick
FINNEGAN, HENDERSON, FARABOW,
GARRETT & DUNNER, L.L.P.
901 New York Avenue, N.W.
Washington, D.C. 20001

/s/ Karen Jacobs Louden

Karen Jacobs Louden (#2881)

TAB 1

US005142622A

United States Patent [19]

Owens

[11] Patent Number: 5,142,622

[45] Date of Patent: Aug. 25, 1992

[54] SYSTEM FOR INTERCONNECTING APPLICATIONS ACROSS DIFFERENT NETWORKS OF DATA PROCESSING SYSTEMS BY MAPPING PROTOCOLS ACROSS DIFFERENT NETWORK DOMAINS

[75] Inventor: **Gary L. Owens**, Mountain View, Calif.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **304,696**

[22] Filed: **Jan. 31, 1989**

[51] Int. Cl.[5] **G06F 13/12**

[52] U.S. Cl. **395/200**; 364/280.9; 364/284.3; 364/284.4; 364/284; 364/DIG. 1; 395/700; 395/500

[58] Field of Search .... 395/200, 700, 500; 364/200 MS File, 900 MS File

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,274,139	6/1981	Hodgkinson	364/200
4,322,792	3/1982	Baun	364/200
4,415,986	11/1983	Chadra	364/900
4,500,960	2/1985	Babecki	364/200
4,530,051	7/1985	Johnson	364/200
4,575,793	3/1986	Morel	364/200
4,586,134	4/1986	Norstedt	364/200
4,604,686	8/1986	Reiter	364/200
4,612,416	9/1986	Emerson	379/88
4,631,666	12/1986	Harris	364/200
4,677,588	6/1987	Benjamin et al.	364/900
4,679,189	7/1987	Olson	370/60
4,703,475	10/1987	Dretzka et al.	370/60
4,706,081	11/1987	Hart	340/825.03
4,736,369	4/1988	Barzilai	370/94.1
4,760,395	7/1988	Katzeff	370/60
4,768,150	8/1988	Chang et al.	364/300
4,790,003	12/1988	Kepley	379/88
4,811,216	3/1989	Bishop	364/200
4,825,354	4/1989	Agrawal	364/200
4,831,518	5/1989	Yu	364/200
4,849,877	7/1989	Bishop	364/200
4,855,906	8/1989	Burke	364/200
4,882,674	11/1989	Quint	364/900
4,893,307	1/1990	McKay	370/60
4,901,231	2/1990	Bishop	364/200

OTHER PUBLICATIONS

Introducing the UNIX System by H. McGilton and R. Morgan pp. 1-7 McGraw-Hill Software Series.

The UNIX Book by Mike Banahan and A. Rutter, J. Wiley & Sons Inc. 1983 pp. 107-120.

The Design of the UNIX Operating System, by Bach, M. J., 1986, Prentice Hall, Englewood Cliffs, N.J.

Dictionary of Computing, 8th Ed., Mar. 1987, IBM Document Composition Facility pp. 238, 326.

Data Communications, vol. 16, No. 5, May 1987, New York, US, pp. 120-142, "SNA to OSI: IBM Building Upper-Layer Gateways" by Thomas J. Routt.

The Sixth International Phoenix Conference on Computers and Communications Feb. 25, 1987, Scottsdale, Ariz., USA, pp. 354-360 by R. Martinez et al.

IEEE Micro, vol. 5, No. 2, Apr. 1985, New York, US, pp. 53-66, "A Multimicrocomputer-based Structure for Computer Networking" by A. Faro et al.

IEEE Infocom '87, Proceedings, Sixth Annual Conference, Mar. 31, 1987, San Francisco, Calif., USA, pp. 1045-1052, "Gateways for the OSI Transport Service" by G. V. Bochman et al.

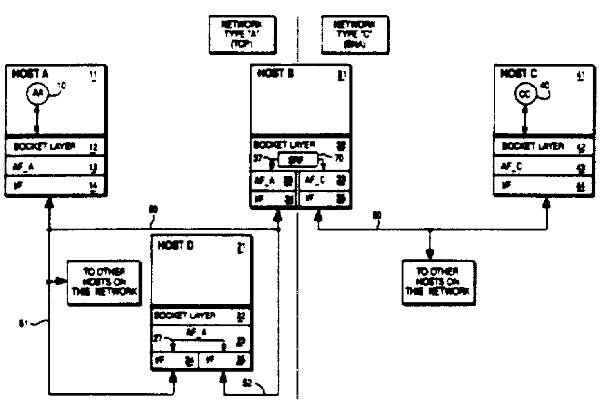
Proceedings, Ninth Data Communications Symposium, Sep. 10, 1985, Whistler Mountain British Columbia, pp. 2-8 "Development of a TCP/IP for the IBM/370" by R. K. Brandiff et al.

Primary Examiner—Thomas C. Lee
Assistant Examiner—Eric Coleman
Attorney, Agent, or Firm—Wayne P. Bailey; Marilyn D. Smith

[57] **ABSTRACT**

The system and method of this invention automatically routes a connection between data processing systems in different network domains. As an example, an application running on a data processing system utilizing a network domain such as TCP (Transmission Control Protocol), can automatically make a connection to another data processing system utilizing a different network domain such as SNA (Systems Network Architecture). The connection is automatically performed in the layer containing the communication end point objects. In a preferred embodiment, the connection is automatically performed in the socket layer of the AIX operating system, or in the socket layer of other operating systems based upon the Berkeley version of the UNIX operating system.

8 Claims, 7 Drawing Sheets



TCORDEL0000001

A01

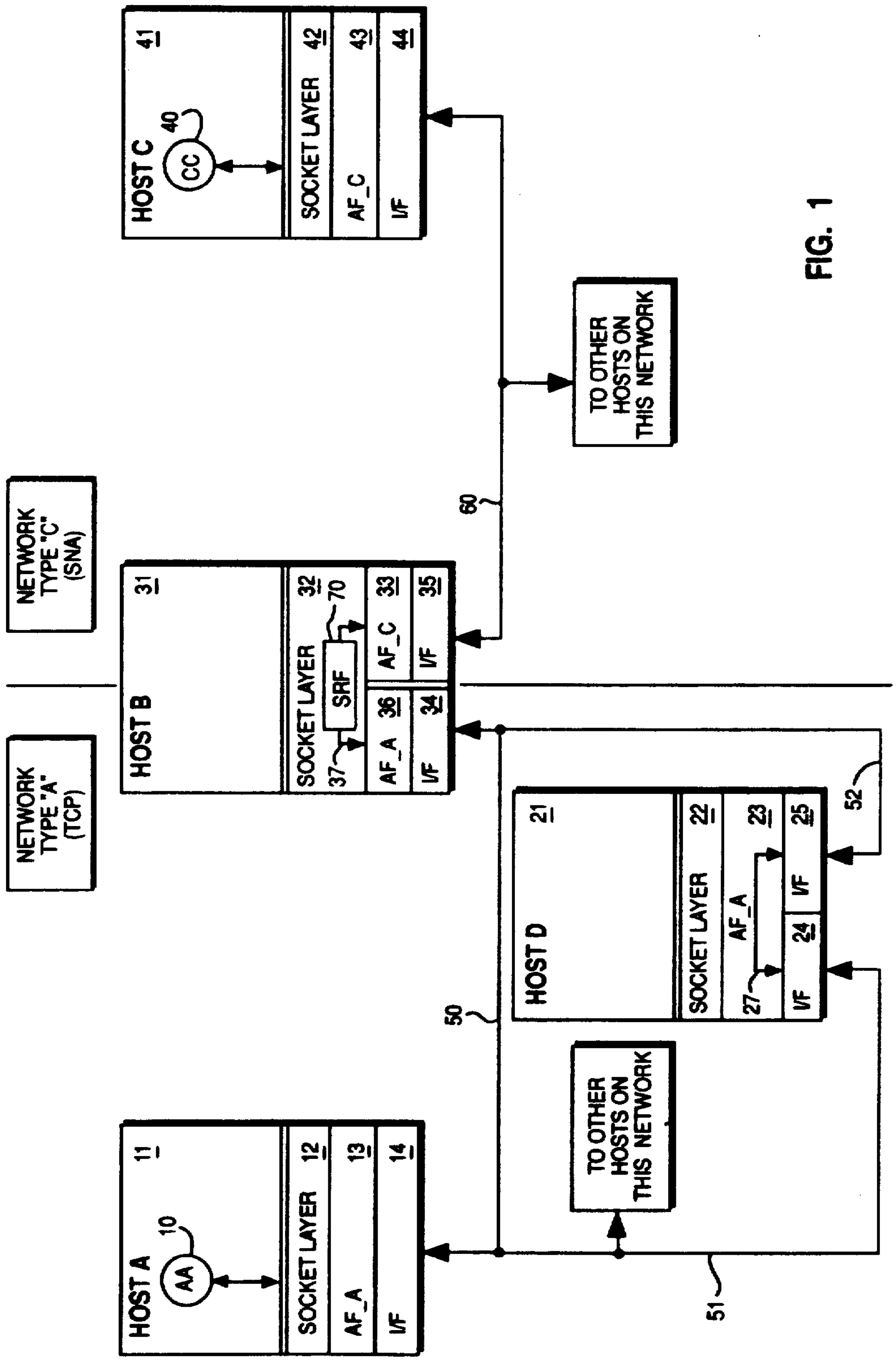
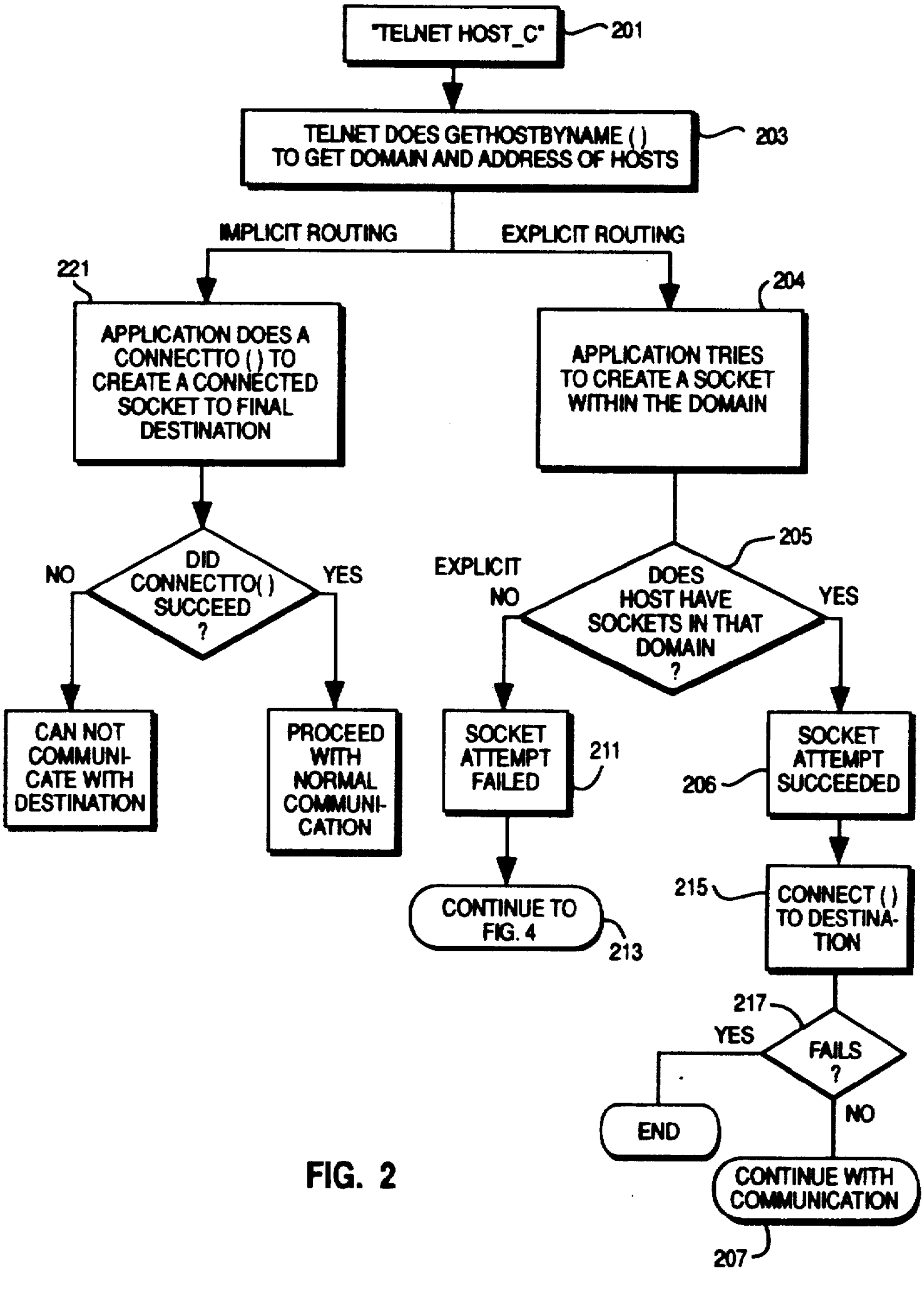

NETWORK TYPE "A" (TCP)
NETWORK TYPE "C" (SNA)
HOST A
AA
10
11
SOCKET LAYER
12
AF_A
13
I/F
14
HOST B
31
SOCKET LAYER
32
SRF
70
37
AF_A
36
AF_C
33
I/F
34
I/F
35
HOST C
CC
40
41
SOCKET LAYER
42
AF_C
43
I/F
44
HOST D
21
SOCKET LAYER
22
AF_A
23
27
I/F
24
I/F
25
TO OTHER HOSTS ON THIS NETWORK
TO OTHER HOSTS ON THIS NETWORK
50
51
52
60

FIG. 1



"TELNET HOST_C"
201
TELNET DOES GETHOSTBYNAME ( ) TO GET DOMAIN AND ADDRESS OF HOSTS
203
IMPLICIT ROUTING
EXPLICIT ROUTING
221
APPLICATION DOES A CONNECTTO ( ) TO CREATE A CONNECTED SOCKET TO FINAL DESTINATION
204
APPLICATION TRIES TO CREATE A SOCKET WITHIN THE DOMAIN
NO
DID CONNECTTO( ) SUCCEED ?
YES
EXPLICIT
NO
205
DOES HOST HAVE SOCKETS IN THAT DOMAIN ?
YES
CAN NOT COMMUNI-CATE WITH DESTINATION
PROCEED WITH NORMAL COMMUNI-CATION
SOCKET ATTEMPT FAILED
211
206
SOCKET ATTEMPT SUCCEEDED
CONTINUE TO FIG. 4
213
215
CONNECT ( ) TO DESTINA-TION
217
YES
FAILS ?
NO
END
CONTINUE WITH COMMUNICATION
207

FIG. 2

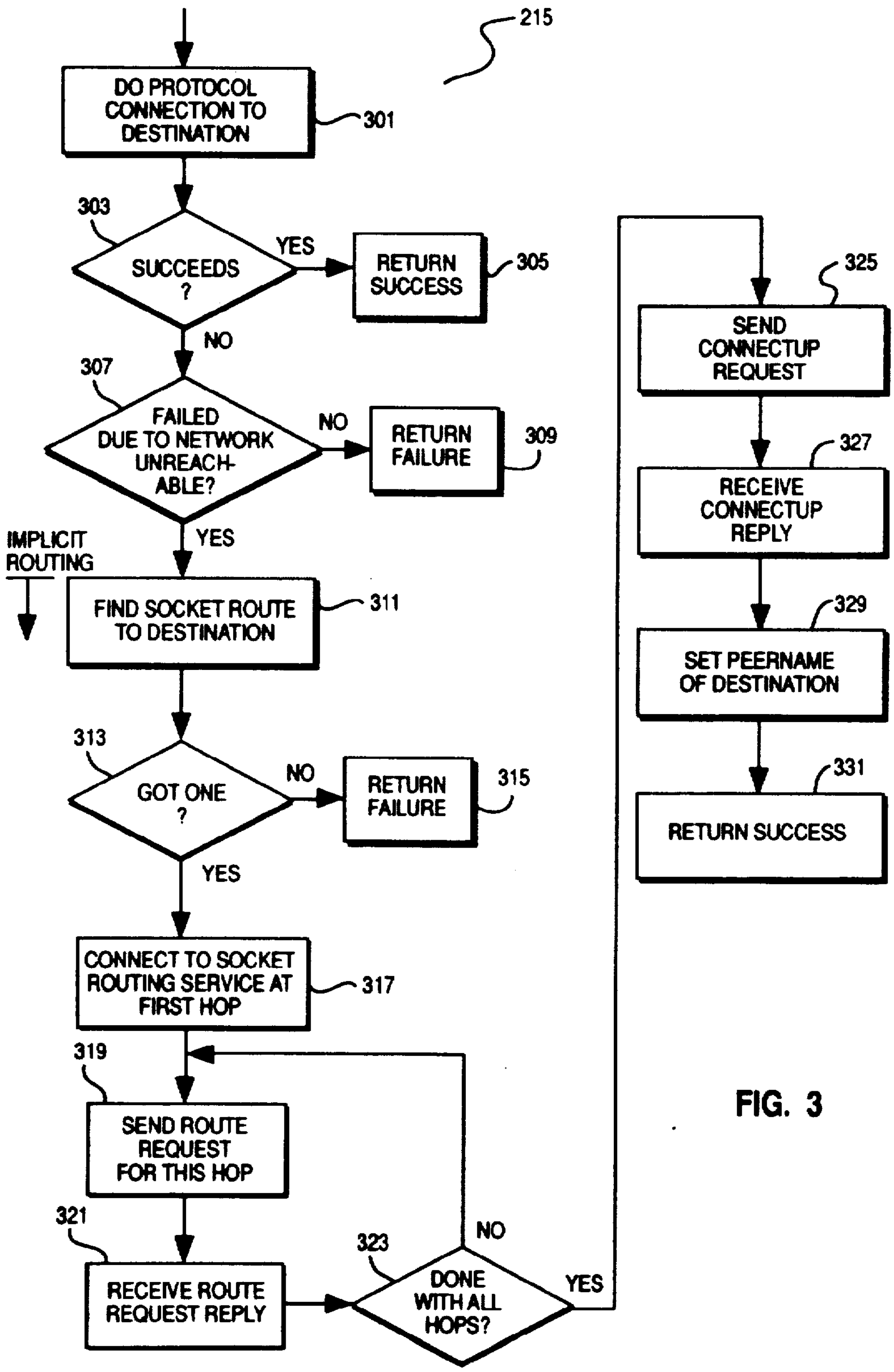

215
DO PROTOCOL CONNECTION TO DESTINATION
301
303
SUCCEEDS ?
YES
RETURN SUCCESS
305
NO
307
FAILED DUE TO NETWORK UNREACH-ABLE?
NO
RETURN FAILURE
309
IMPLICIT ROUTING
YES
FIND SOCKET ROUTE TO DESTINATION
311
313
GOT ONE ?
NO
RETURN FAILURE
315
YES
CONNECT TO SOCKET ROUTING SERVICE AT FIRST HOP
317
319
SEND ROUTE REQUEST FOR THIS HOP
321
RECEIVE ROUTE REQUEST REPLY
323
DONE WITH ALL HOPS?
NO
YES
325
SEND CONNECTUP REQUEST
327
RECEIVE CONNECTUP REPLY
329
SET PEERNAME OF DESTINATION
331
RETURN SUCCESS

FIG. 3

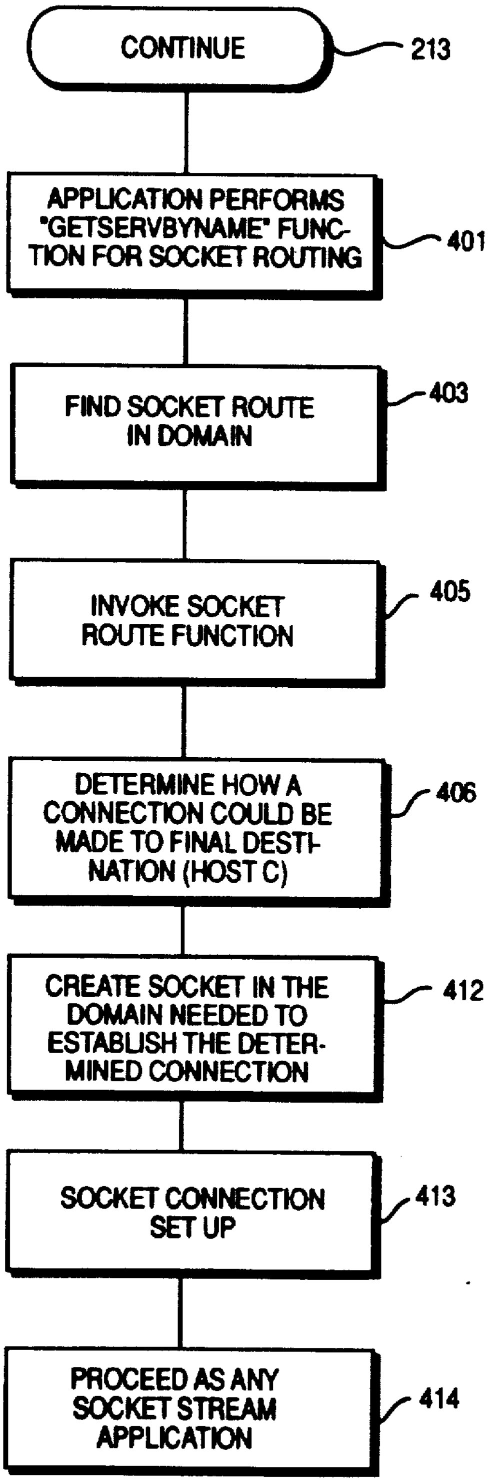

CONTINUE
213
APPLICATION PERFORMS "GETSERVBYNAME" FUNCTION FOR SOCKET ROUTING
401
FIND SOCKET ROUTE IN DOMAIN
403
INVOKE SOCKET ROUTE FUNCTION
405
DETERMINE HOW A CONNECTION COULD BE MADE TO FINAL DESTINATION (HOST C)
406
CREATE SOCKET IN THE DOMAIN NEEDED TO ESTABLISH THE DETERMINED CONNECTION
412
SOCKET CONNECTION SET UP
413
PROCEED AS ANY SOCKET STREAM APPLICATION
414

FIG. 4

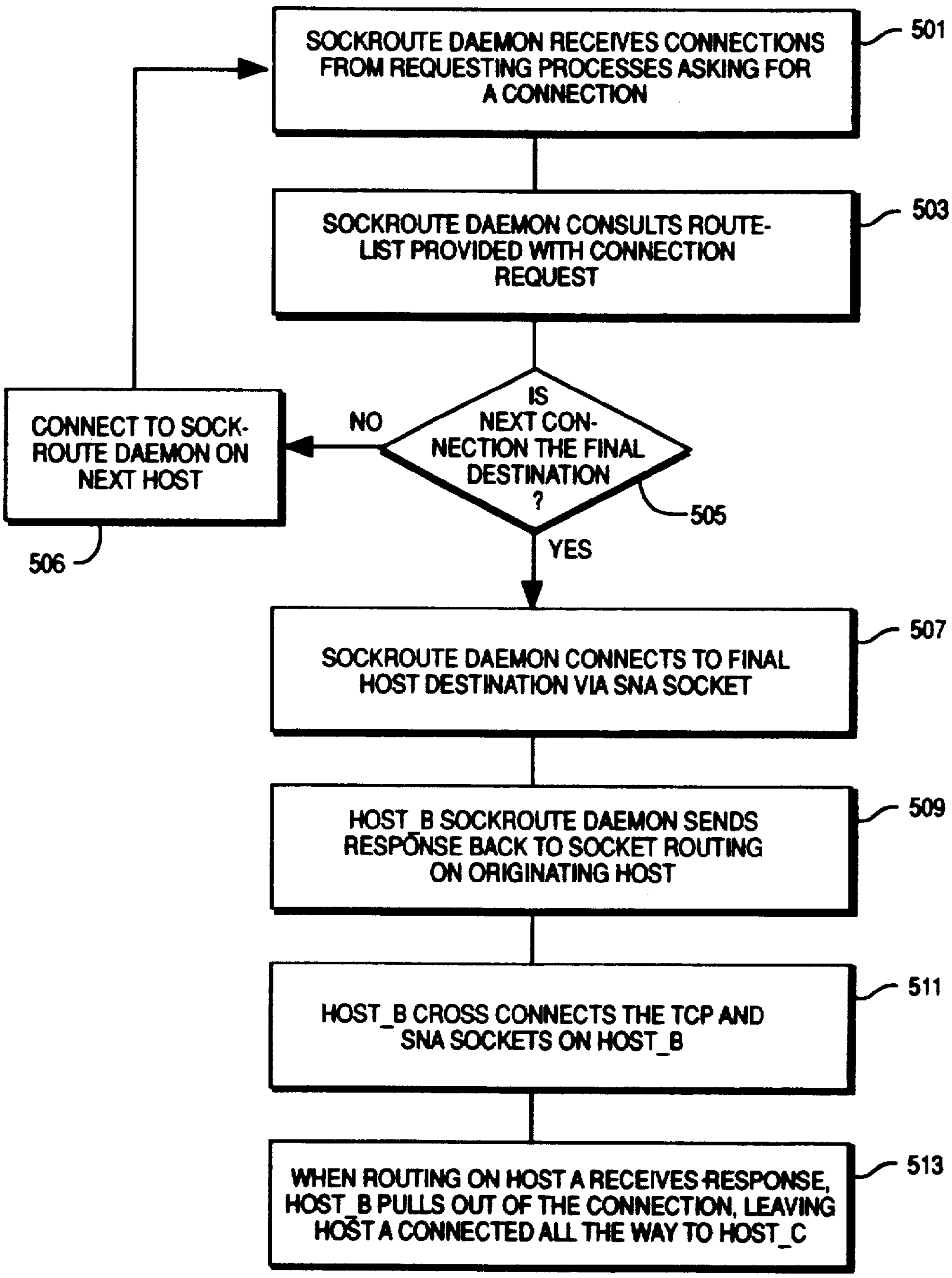
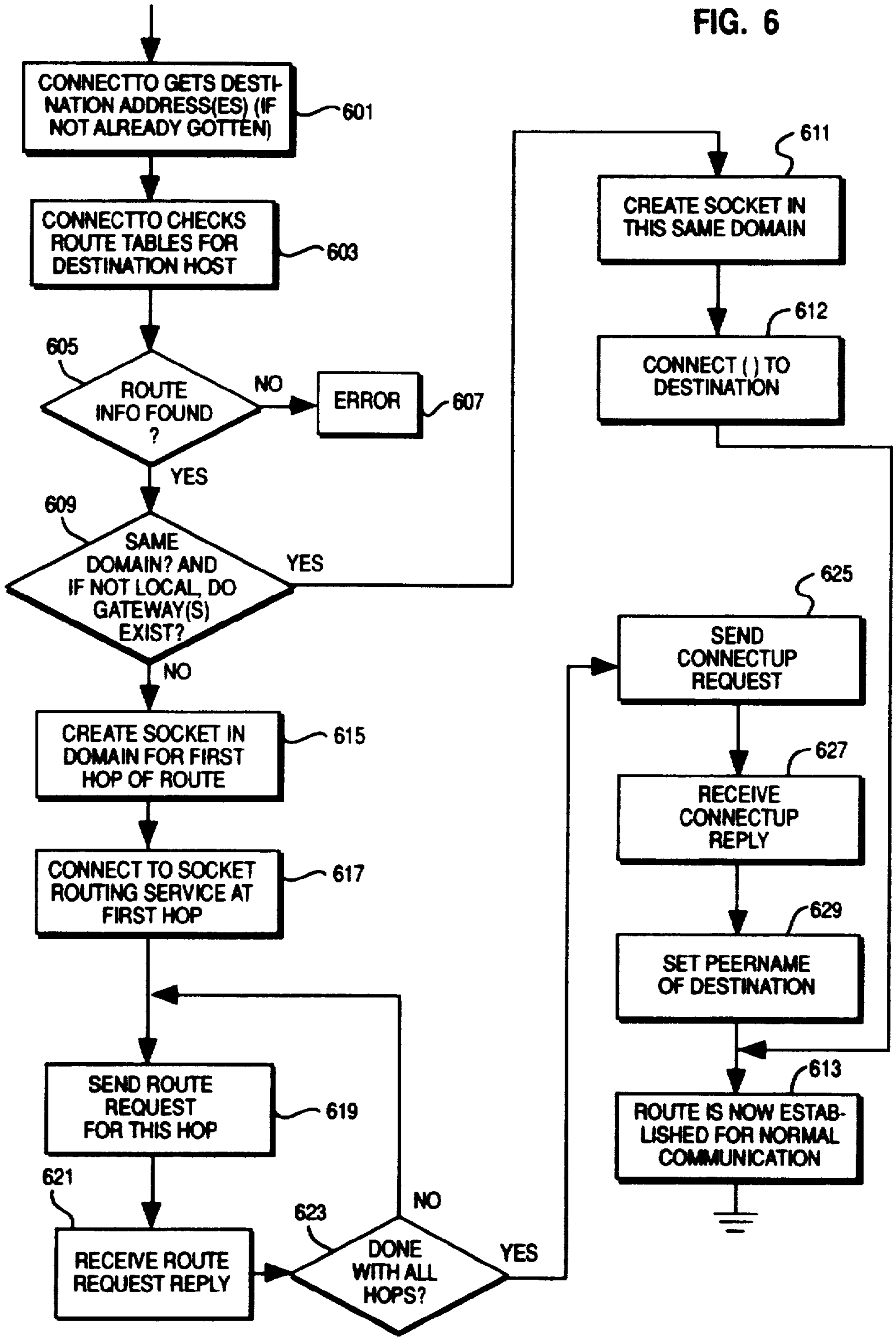

SOCKROUTE DAEMON RECEIVES CONNECTIONS FROM REQUESTING PROCESSES ASKING FOR A CONNECTION
501
SOCKROUTE DAEMON CONSULTS ROUTE-LIST PROVIDED WITH CONNECTION REQUEST
503
IS NEXT CONNECTION THE FINAL DESTINATION ?
505
NO
CONNECT TO SOCKROUTE DAEMON ON NEXT HOST
506
YES
SOCKROUTE DAEMON CONNECTS TO FINAL HOST DESTINATION VIA SNA SOCKET
507
HOST_B SOCKROUTE DAEMON SENDS RESPONSE BACK TO SOCKET ROUTING ON ORIGINATING HOST
509
HOST_B CROSS CONNECTS THE TCP AND SNA SOCKETS ON HOST_B
511
WHEN ROUTING ON HOST A RECEIVES RESPONSE, HOST_B PULLS OUT OF THE CONNECTION, LEAVING HOST A CONNECTED ALL THE WAY TO HOST_C
513

FIG. 5

FIG. 6


CONNECTTO GETS DESTINATION ADDRESS(ES) (IF NOT ALREADY GOTTEN)
601
CONNECTTO CHECKS ROUTE TABLES FOR DESTINATION HOST
603
605
ROUTE INFO FOUND ?
NO
ERROR
607
YES
609
SAME DOMAIN? AND IF NOT LOCAL, DO GATEWAY(S) EXIST?
YES
NO
CREATE SOCKET IN DOMAIN FOR FIRST HOP OF ROUTE
615
CONNECT TO SOCKET ROUTING SERVICE AT FIRST HOP
617
SEND ROUTE REQUEST FOR THIS HOP
619
621
RECEIVE ROUTE REQUEST REPLY
623
DONE WITH ALL HOPS?
NO
YES
611
CREATE SOCKET IN THIS SAME DOMAIN
612
CONNECT ( ) TO DESTINATION
625
SEND CONNECTUP REQUEST
627
RECEIVE CONNECTUP REPLY
629
SET PEERNAME OF DESTINATION
613
ROUTE IS NOW ESTABLISHED FOR NORMAL COMMUNICATION

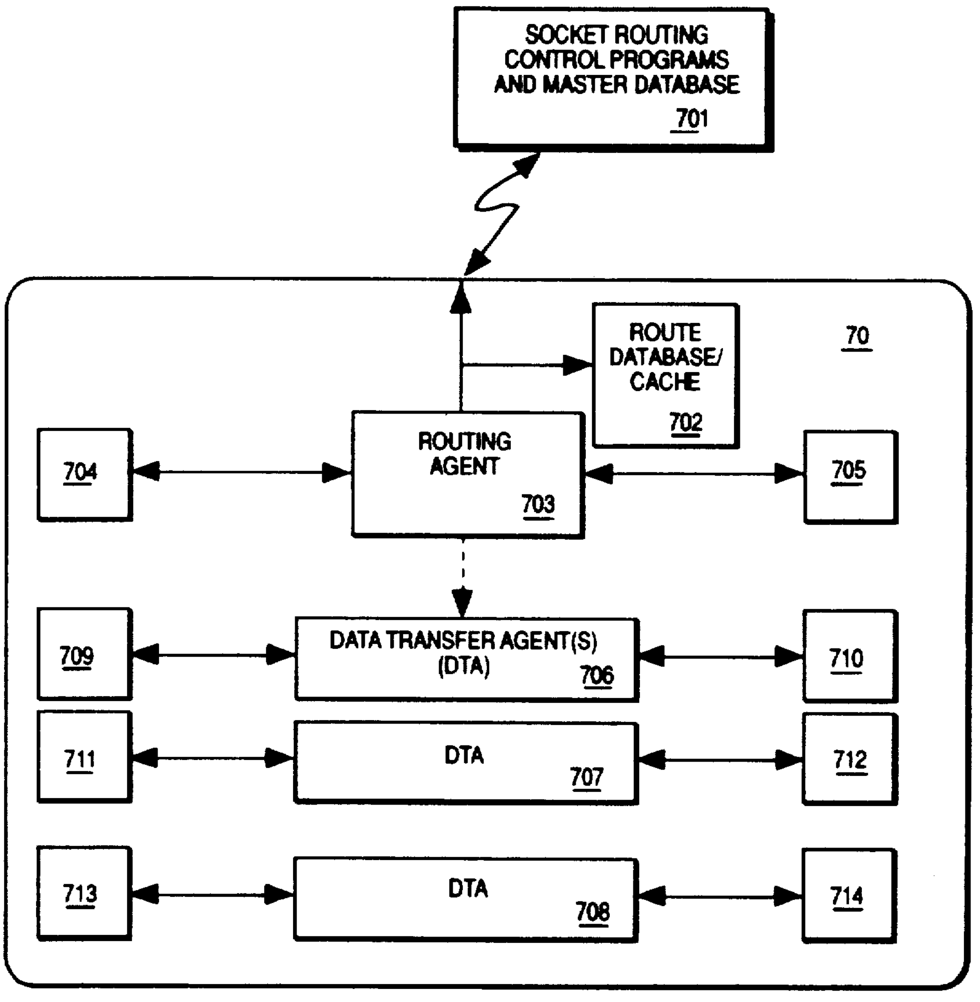

SOCKET ROUTING CONTROL PROGRAMS AND MASTER DATABASE
701
ROUTE DATABASE/ CACHE
702
70
ROUTING AGENT
703
704
705
DATA TRANSFER AGENT(S) (DTA)
706
709
710
711
DTA
707
712
713
DTA
708
714

FIG. 7

SYSTEM FOR INTERCONNECTING APPLICATIONS ACROSS DIFFERENT NETWORKS OF DATA PROCESSING SYSTEMS BY MAPPING PROTOCOLS ACROSS DIFFERENT NETWORK DOMAINS

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to a network of data processing systems, and more specifically to the interconnection of a plurality of data processing systems between different network protocol domains, such as the different network protocol domains of SNA and TCP/IP.

2. Description of the Related Art

A system having multiple domains has at least one data processing system that is interconnected to at least two different data processing systems through at least two different network domains, i.e. network protocol architectures. A problem with multiple domains is the difficulty in allowing communication between machines which are connected to another type of network. For example, a data processing system utilizing SNA LU 6.2 as its network protocol can not automatically communicate with another data processing system utilizing TCP/IP as its network protocol. Both SNA LU 6.2 and TCP/IP are examples of stream protocols where data flows as a stream of indeterminate lengths, and the bytes are delivered in the correct order. The problem is routing a stream of bytes from a data processing system that utilizes a reasonably equivalent protocol, such as a stream protocol, to another data processing system that also utilizes a reasonable equivalent protocol, such as the stream protocol of this example, but wherein the two protocols are not the exact same protocol, such as SNA LU 6.2 and TCP/IP.

It is known to solve the above problem at the application program level. An application program which is running on a data processing system at one end of the connection may be designed to utilize a specific network protocol. In this case, it is known to modify the application in order to reimplement the application to work over another protocol. This requires changing the source program code of the original application by some amount. Depending upon how the application program was originally designed, this may require a substantial amount of changes to the program code.

It is also known to solve the above problem by implementing the same protocol on both machines. For example, in order to use an SNA transaction application running in an SNA network, to apply transactions against data processing systems utilizing a TCP network, one could reimplement that transaction application against TCP by then putting TCP on the client data processing system, put IP over SNA, and gateway between the two. The client data processing system can then be implemented utilizing TCP/IP. The problem with this approach is having to reimplement the application to utilize the different protocol at one end of the network or the other. This is especially burdensome if the application is large and complex.

There are some application level protocols that handshake back and forth over SNA, e.g. 3270 SNA. These have their own data format with meta-data in the data stream. There are other application level protocols, such as Telnet over TCP, that talk back and forth that have meta-data and data in the data stream. However, one can not get these two to talk together since these two have different data and meta-data in their data streams.

If an application utilized one protocol, and that application were to run on a data processing having a different protocol, knowing the data stream format, one could write the client half of the application on the data processing system utilizing the other protocol.

Therefore, in order to extend network connectivity, it is known to reimplement the application to utilize the different protocol, put one protocol on top of the other, and gateway between the two. It is also known to build a larger network utilizing each type of protocol through replication and duplication.

The term "sockets" is an application program interface (API) that was developed for the Berkeley version of AT&T's UNIX[1] operating system for interconnecting applications running on data processing systems in a network. The term socket is used to define an object that identifies a communication end point in a network. A socket can be connected to other sockets. Data can go into a socket via the underlying protocol of the socket, and be directed to appear at another socket. A socket hides the protocol of the network architecture beneath a lower layer. This lower layer may be a stream connection model (virtual circuit), or a datagram model (packet), or another model.

[1]UNIX is licensed and developed by AT&T. UNIX is a registered trademark of AT&T in the U.S.A. and other countries.

A stream connection model refers to a data transmission in which the bytes of data are not separated by any record or marker. A virtual circuit implies that there appears to be one communications end point connected to one other communications endpoint. When the connection is established, only those two end points can communicate with each other.

Sockets are typed by domain (address family or network type), and model type (stream, datagram, etc.). If needed, the socket can be further specified by protocol type or subtype. The domain specifies the addressing concept utilized. For example, there is an internet IP domain, and also a SNA domain for networks utilizing TCP and SNA, respectively. As used herein, the word "domain" is used to refer to the address family of a socket, and not to a domain-naming domain. A domain-naming domain is a concept of a related group of hierarchical addresses, wherein each part of the address is separated by a delimiter, such as a period.

Since a socket is specified by the domain, sockets do not allow cross domain connections. This means that if an application program creates a socket in the Internet (Darpa) domain, it can only connect to sockets in that same domain. Note: "Darpa" is used to specify that Internet, short for internetworking, is not only used herein both to generically specify the internet layer of a particular protocol family which contains means for forwarding, routing control, and congestion control, etc., but also as a name for a particular implementation of an internet called the Internet or the Darpa Internet, or the Arpa Internet. Another name for this internet

layer is the Internet Protocol (IP). TCP/IP is also commonly used to refer to this protocol.

Originally, the requirement that a socket can only connect to sockets in the same domain was a reasonable restriction. This simplified the program code when there was only one really useful domain anyway. With the advent of the usage of other domains (specifically SNA), cross domain connections have become desirable. For example, cross domain connections would allow mailers to transport mail among domains. Also, cross domain connections would allow programs to communicate using the existing communication networks.

SUMMARY OF THE INVENTION

It is therefore an object of this invention to automatically route connections between data processing systems that utilize different protocols, independently of said applications running on said data processing systems.

It is a further object of this invention to route, at the socket level, between two networks when a cross-domain connection attempt is detected.

It is a further object of this invention to facilitate the interconnection between data processing systems by allowing socket based applications to easily span across different networks.

It is a further object of this invention to communicate between data processing systems in which one of the data processing systems utilizes TCP/IP and the other data processing system utilizes SNA.

It is a further object of this invention to communicate between two data processing systems via a third data processing system utilized as a TCP to SNA gateway.

It is a further object of this invention to communicate through a connection between two data processing systems both utilizing TCP on each of their local Internets, by bridging the network connection with a long haul SNA connection.

The system and method of this invention automatically routes a connection between data processing systems, independently of an application running on the data processing systems, having different network domains. The preferred embodiment describes the cross domain interconnections with reference to the different network domains of TCP (transmission control protocol) and SNA (systems network architecture).

The routing is automatically performed at a layer which contains the communication end point objects. In the AIX[2] operating system, and other operating systems based upon the Berkeley version of the UNIX operating system, this layer is called the socket layer.

[2] Trademark of IBM Corporation

An intermediate processing system is utilized to gateway between a processing system utilizing a network domain such as TCP, and another processing system utilizing a different network domain such as SNA. Alternatively, the client data processing system can be implemented utilizing TCP/IP which can then be gatewayed through socket routing on the same machine into an SNA data stream without an intermediate processing system performing the socket routing.

In any event, the socket layer which performs the socket routing contains facilities to automatically route a connection across different domains.

In the client processing system which is attempting to create a connection, a socket is created in a particular domain. If the socket is in a different domain, the socket does not fail if the socket routing facility of this invention is implemented. The connect function is modified to catch the attempts at a cross domain connection. If a connect function is attempted on a socket in a different domain, then the socket routing facility of this invention is invoked.

Alternatively, a connectto function can be implemented which takes the place of and combines the functions of the socket function and the connect function. With the connectto function, a socket is not created until the route is known. This alleviates the unnecessary work of creating a socket which may fail, and then performing actions as a result of the failed socket. The connectto function determines how a connection can be made, and then creates a socket in the domain that is needed to establish the determined connection.

Through either of the above approaches, a connection to a socket in a different domain can be made through an intermediate socket. When data arrives from one end of the connection to the intermediate socket, the intermediate socket immediately sends the data to the other end of the connection instead of queuing the data for process intervention at the intermediate processing system.

In addition, if the intermediate socket is queried for the address of the other end of the connection, the intermediate socket identifies the connecting host as opposed to the intermediate host. In this way, the socket routing facility of the intermediate host is transparent to the hosts at each end of the connection.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a diagram showing a connection from a process AA on host A to a process CC on host C. Socket routing is utilized to cross the boundary between the networks of type A and type C at host B.

FIG. 2 is a flow diagram showing the operational scenario of FIG. 1 using explicit and implicit routing.

FIG. 3 is a flow diagram showing the modified steps in performing a connect ( ) function to a destination.

FIG. 4 is a flow diagram showing the steps of creating a socket if the host does not have a socket in the specified domain.

FIG. 5 is a flow diagram showing the steps performed at host B.

FIG. 6 is a flow diagram showing the steps of a connectto ( ) function.

FIG. 7 is a more detailed diagram of the socket routing facility of this invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

The following description describes an architecture for routing virtual circuits based on sockets. Although this implies stream sockets, the invention is not limited to stream protocols or to sockets. The concepts of this invention could be applied to similar communication end points that are utilized within other operating systems.

Referring to FIG. 1, a process AA, **10**, in a data processing system **11**, host A, desires to connect its socket facilities **12** to the process CC, **40**, in a data processing system **41**, host C. The data processing system **11** is shown as only supporting a particular domain of sockets AF_A, **13**, such as TCP, and data processing system **41** is shown as only supporting sockets that exist in the domain having address family C, **43**. Since the naming conventions and the underlying transport mechanisms

are different between address family A, 13, and address family C, 43, no interconnection can take place without an intermediate facility. The intermediate facility is the socket routing facility 70 in socket layer 32, which exists in data processing system 31, shown as host B.

To describe the initiation of a connection, the process AA, 10, in the data processing system 11, will activate a connection through the sockets programming interface to the general socket code, 12, which in turn goes through the address family specific socket code for AF_A, 13. The necessary data and control information will be handled by the interface and physical access layers, 14. The data will then go out on the network 50 and end up going into data processing system 31, shown as host B, via the interface layer 34, and then through the code for address family 30 A, shown as AF_A, 36.

For comparison, data processing system 21, shown as host D, shows existing internet routing within a single address family, the address family A, AF_A, 23. It should be noted that the cross connection occurs within the address family A, 23. Almost any TCP/IP implementation can route within its own address family. Likewise, SNA has similar gateway and forwarding capabilities. The cross over as shown in data processing system 21 is independent of the model type of either stream or datagram. It is only dependent upon being within the same network domain.

In data processing system 31, the connection request packets will go through the interface layer code 34 to the address family A code, AF_A, 36, through the general socket layer 32, and into the socket routing code 70. The socket routing code facility 70, is where the address mapping and cross connection takes place. The cross connection arrows 37 are shown drawn in the socket routing layer 70 of data processing system 31, as opposed to the cross connection arrows 27 which are shown in the address family code 23 of data processing system 21.

A connection request generated in the socket routing code 70 of data processing system 31 will then go down through the address family C code, AF_C, 33, and through the interface layer code 35 for the other network 60, such as SNA. The connection request packets go across the network 60 to the interface layer code 44, up to the address family C code, AF_C, 43, continuing through the general socket interface layer code 42 where the connection is registered. Then the process CC, 40, can respond to the connection request in order to establish the connection between cross domain networks.

FIG. 7 shows item 70 of FIG. 1 in greater detail. Item 701 is the programs and data for controlling the socket routing facility. A connection request to establish socket routing will come in on the sockets for this service, items 704, and 705. The routing agent software, item 703, will accept the connection, which creates a data socket, items 709-714. The route request message will come in on that data socket, and the routing agent, 703, will consult its route database, 702, to see if a route is possible. If a route is possible, the routing agent, 703, will consult its route database, 702, on how to establish the route. Then, the routing agent creates a matching data socket (item 710 for item 709, etc.), and connects to the next hop. When the routing agent software receives any replies for further route hops, it forwards them back to the socket routing requestor via the accepted data socket. When all hops are made, the socket routing agent will create a data transfer agent, items 706-708, that joins the pairs of data sockets, and forwards data from one to the other and vice versa.

The above scenario is further described in the following programming design language code. The following includes examples and uses programs and function names to describe the operational scenario of FIG. 1. The following operational scenario assumes a telnet (or similar program) connected to a remote processing system that is separated by at least one domain boundary. The following uses three machines: "host_A" is connected to "host_B" via TCP, and "host_B" is connected to "host_C" via SNA.

```
*/from application view/*
user on host_A says "telnet host_C"
telnet does a gethostbyname for "host_C"
telnet tries to create a socket for domain of "host_C"
        - it fails
telnet does a getservbyname for sockroute
        - it finds (the only) sockroute available in TCP
        domain
telnet invokes sockroute function to get which domain
        to initiate the connection in (or to get a route
        to host_C)
since telnet knows it is now using socket routing it
        uses the (initial domain and routelist) to
        1. create a socket in its initial domain. (TCP)
        2. connects to sockaddr of "host_C" telnetd
                -or "connectto routelist telnetd"
when socket connect succeeds, proceed as any
        SOCK_STREAM app would
- alternatively ( with connectto() as "full function")
user on host_A says "telnet host_C"
telnet does a gethostbyname (or getaddrbyname) for
        "host_C" - to see if it exists and to get
        host_C's address
telnet does a "connectto ( host_C:telnetd,
        SOCK_STREAM) - which gets a connected socket.
```

COPYRIGHT IBM CORPORATION 1988

The above program design language code is further explained with reference to FIGS. 2-4. The term "telnet" is a remote terminal emulator having the argument "host_C". This invokes the terminal emulator to a remote host, which in this case is "host C", step 201, FIG. 2. "Gethostbyname" is a function call of the telnet program which gets the addressing information for host C, step 203, FIG. 2. The addressing information for host_C will include a domain and an address within the domain.

At this point, the routing can be performed either explicitly or implicitly. Explicit action would involve the user code invoking a router function, if the initial attempt to create a socket fails. Implicit action would simply be doing a connectto ( ) on the destination address. In explicit routing, the advantage is explicit control by the application. The disadvantages are lack of centralized control, and more complicated user code. In implicit routing, the advantages and disadvantages are just the opposite of those stated above. In implicit routing, the advantages are more centralized control, and less complicated user code. In implicit routing, the disadvantage is that the application does not have direct control.

With explicit routing, Telnet tries to create a socket within that domain, step 204. If the host does not have sockets of that domain, step 205, the socket creation will fail, step 211. At this point, the application, Telnet, invokes a router function, step 213 FIG. 4, if the socket attempt failed, step 211. If the host does have sockets within this domain, the socket attempt will succeed, step 206. If the socket attempt succeeds, the application

does a connect ( ), step 215. The connect ( ) is further shown with reference to FIG. 3. If the connect ( ) succeeds, step 217, FIG. 2, the communication between the two processes proceeds as is typically known in the art, step 207.

If a connect in the same socket domain failed, then (possibly with a socket option set) the socket routing would be invoked. This provides implicit routing, FIG. 3, even in the case of a connection between two domains of the same type, using an intermediate domain of different type.

As shown in FIG. 3. modifying the function connect ( ) enables the connect ( ) to catch those situations in which socket routing is needed to gateway between two like domains using unlike domains. If a normal connection, step 301, fails, step 303, and the failure is due to the destination network being unreachable, step 307, then an attempt at implicit routing will be made. This begins with step 311 where a socket route is sought for the destination. If no route is found, then an error is reported, step 315. If a route is found, a connection is made to the socket routing service at the first hop, step 317. Then, a route request is sent, step 319, and the route request replies are received, step 321, until all the hops are connected, step 323. At this time, a connect up request is sent to tell all of the routers to set up the line for data transmission, step 325. After the connect up reply is received, step 327, the peer address of the destination is set for the local socket, step 329, and an indication of success is returned to the invoker of this connect, step 331.

Referring back to FIG. 2, a connectto function can be added to the generic socket layer code to implement implicit routing from an application level, step 221, FIG. 2. The connectto function is called instead of a socket function and a connect function. The function of the socket system call and the function of the connect are combined into the connectto function. The advantage of this is that the connectto function can handle more addressing issues. Also the connectto function does not need to create a socket in the kernel, which may fail, and then have to act upon the failed socket.

The socket parameters of the connectto function would include the type and the protocol. Since the previous connect call has arguments for the host name, the connectto function would take the name of the host in a more portable form, such as the name of the host in a text stream, whereas, connect takes the name of the host in a socket structure.

Referring to FIG. 6, the connectto( ) function is further described. If connectto( ) is implemented so that it takes a host name as an argument, then it gets the destination address, step 601. Using this address, the function checks the route table for the destination, step 603. If no route is found, step 605, then an error is returned, step 607. If the destination is in the same domain, and no unlike domains are required for gateways, step 609, then a socket is created in the same domain, step 611. A normal connection is established to the destination, step, 612. The route for communication is then established, step 613.

If the destination is not the same domain or unlike domains are required for gateways, step 609, then a socket is created in the domain of the first hop, step 615. A connection to the socket routing service at the first hop is then established, step 617. A route request is sent, step 619, and a reply to the request is received, step 621, until all hops are connected, step 623. After this, a connect up request is sent, step 625, and its reply is received, step 627. The peername of the destination is set for the local socket, step 629. The route is now available for normal communications, step 613.

With the following modifications, referred to as socket routing, the creation of a socket can continue, step 213, as shown in FIG. 4, when the host does not have a socket in the specified domain, step 205, FIG. 2. The modifications take place at the client side, host_A. Host_C is referred to as the server.

The telnet application performs a "getservbyname" function for the socket routing service, step 401, FIG. 4. If, for example, the host only has sockets in the TCP domain, telnet will find the only socket route available in the TCP domain, step 403. Next, telnet uses the sockroute function, step 405, to determine the route and what domain of socket to create, step 406. Then, the socket is created for the initial hop of the route, step 412, and then the connection would be set up, step 413. At this point, the application can talk to the host as it otherwise would have with any other socket stream, and in this case, using the telnet data stream, step 414.

Assuming the route initialization is done by a daemon or library function on host_A (and not kernel code), then host_A's socket code doesn't really have much to do with socket routing. Basically, if socket routing is performed outside of the operating system kernel on host_A, then no changes to host_A's socket code need to be made.

The following programming design language code, and the following description with reference to FIG. 5 describes what happens on host_B.

```
*/ on host_B /*
sockroute daemon receive connection from host_A
    (asking for connection to host_C)
sockroute daemon consults route table -or route list
    provided with connection request.
sockroute daemon decides to connectto to host_C via
SNA socket
    (since it is last hop, it doesn't need to connect
    to a sockroute daemon on host_C)
when connection completes, host_B sockroute daemon
    1. sends response back to socket routing on
       host_A
    2. cross connects the TCP and SNA sockets on
       host_B
when routing on host_A receives response, it pulls out
of the way, leaving telnet connected all the way to
host_C
```

COPYRIGHT IBM CORPORATION 1988

Essentially, the above code describes the scenario in which a service waits around for a connection. With reference to FIG. 5, the sockroute daemon, which runs on host_B, receives connections from other processes requesting its services, step 501. The sockroute daemon is analogous to a telephone operator who is requested to make a connection to another person from a caller. The requesting process, caller, supplies the sockroute daemon, operator, with the necessary connection information in order to make the connection, step 503. Once the sockroute daemon makes the connection, the sockroute daemon leaves the connection. If this connection leads to the final destination, step 505, no other sockroute daemons on a next host need to be called, and the sockroute daemon connects to the final host destination via a SNA socket, step 507. However, it is possible to have multiple sockroute daemons, operators, that are needed to make a connection from a first host to a final host

destination. If this connection does not lead to the final host connection, then another sockroute daemon on a next host must be called, step **506**, and the above steps repeated.

The sockroute daemon on host_B then sends a response back to the socket routing service on the originating host, host_A, step **509**. Host_B cross connects the TCP and SNA sockets on host_B, step **511**. When the routing service on host_A receives the response, host_B pulls out of the way. This leaves a telnet connection all the way from host_A to host_B, step **513**.

It should be noted that since host_C is the end of the line, its socket layer is entirely unaffected for data transfer purposes.

There is a function called getpeername ( ) that is part of the sockets programming interface. A socket can also be queried as to which service is connected to it. For example, if host_C queried its socket to determine which service at the other end it was connected to, the response would be the intermediate host, host_B, instead of the actual service at the other end of the connection which in this example is host_A. Therefore, the getpeername would need input from the socket routing code at both ends of the connection, as well as some kernel changes, for it to work in a transparent fashion. For transparency, the getpeername would respond with host_A, the real end of the completed connection, if the socket in host_C was queried as to the party at the other end of the connection.

The details of the address mapping and socket routing facilities within the socket layer **32**, which effectuates the cross domain connections, are described hereafter.

Gatewaying of socket based protocols is achieved by looping two sockets together at the top end. Such a mechanism would allow a router to create a path that would cross domain boundaries. A router in this context would be program code that would decide how to get to one data processing system to the other such as in the internet layer of TCP/IP. SNA also has similar code. The mechanism for looping two sockets together at the top end would not require file descriptors, or process switching time on the connecting node, once the connection is established. The following illustrates the changes to the socket layer interface of an operating system, such as the AIX operating system that utilizes the Berkeley sockets, that may be made to implement socket routing of this invention. These changes include the following:

modify "connect" to catch cross domain connects

add "connectto" to implement implicit routing from application level.

as an option, create library functions for routing

modify socket buffer handling, etc. to allow cross connections without process intervention

as an option, add function so getpeername works transparently

define socket routing protocol and messages (in kernel or as a daemon)

if needed, modify nameserver for domain gateways and routing info.

If connectto is not used to hide the routing from the user in a library, it is also possible to create library functions to perform the routing. However, the user will require a facility to figure out which machine has a socket routing daemon to service an intermediary. These functions(s) would allow a user program to invoke socket routing with minimal effort. Possible function to be defined are:

"get_route"—user program asks for route (useable by connect( ))

"get_type_of_socket_I_should_open_to_get_to_host"—done against the return from "get_route"—or does implicit "get_route".

connectto"—(1) looks up route, (2) creates a socket in proper domain, (3) established connection.

i.e., instead of

```
hp=gethostbyname(host);
(fill in sockaddr from hp . . . )
so=socket(AF_XX, SOCK_STREAM,0); connect (so, sockaddr, sockaddrlen);
```

a program does

```
so=connectto(host, SOCK_STREAM, 0);
```

In addition, modifying socket buffer handling will allow cross domain connections without process intervention. Previously, a socket is set up such that when data arrives, the data is stored in a queue while the data waits for a process to read it. At the gateway, the socket routing machine, when data arrives from one end of the connection, the data has to be automatically sent out the other side to the other end of the connection, and vice versa.

A current implementation of socket buffering would require that a process be running against all the sockets that are cross connected. A more efficient means would be to add this cross connection at a socket buffer layer, so that no process scheduling needs to be done to send the data on its way. In either case, flags are added to the socket data structures.

As previously mentioned, additional function is added to the "getpeername" function to enable the intermediate host to appear transparently in the connection between the originating host destination and the final host destination. Previously, the socket peer address has been handled by protocol dependent means. A change is required so that getpeername( ) works correctly. The change involves having the peer address propagated by the route daemons, in both directions. Then the routing code at each end of the connection would do a "set peer address" operation, which would override the protocol's peer address function.

The socket routing facility of this invention also requires a socket routing protocol and messages. It is desirable that the socket routing code handle routing in a flexible manner. To achieve this, a preferred embodiment of this invention has a socket routing daemon on each machine that is an interdomain gateway. The daemon would be listening on well-known socket(s) for routing requests. When a request came in (via a connecting socket) the routing daemon would examine the request and perform the desired action.

These requests (and their responses) are as follows:

Messages For Socket Routing Protocol

and the information that goes with each message

route request—sent to request a route be set up

originator address

hop destination address

flag for intermediate or final hop

route request reply—received to indicate completion and success/fail of route request

status for success or failure

connectup request—sent to establish normal data pathway

<none>

connectup reply—received to indicate completion and success/fail of connectup request

status for success or failure

The socket routing service code is used to perform routing at the intermediate nodes, i.e. the gateway node. When a request for service arrives at the gateway machine, such as for any other socket connection, the request for service would arrive at a particular socket which would be the socket of the socket routing daemon. The process with this particular socket open could be either in the kernel or running as a user level process.

Therefore, the socket routing service code can be created as a daemon or in the kernel. Preferably, the socket routing service code will exist mostly or completely as a daemon. Some minor parts, such as ioctls (input output controls) to tie sockets together, may exist as part of the kernel. However, these minor parts support the daemon, and are not really a part of the socket routing service code. As an alternative, it is also possible to put the routing implementation part (as opposed to the route figuring out part) in the kernel, which would save process context switch time.

Another modification may be made to implement the socket routing of this invention. The nameserver may be modified for domain gateways and routing information. The (name) domain name server needs to have a type of data for inter(socket) domain gateways. It may also be desirable for it to find gateways when looking up a host address. It would be desirable if it would flag the fact that a host requires an inter(socket) domain gateway to get to it.

While the invention has been particularly shown and described with reference to a preferred embodiment including sockets, the underlying idea of cross domain connections could be achieved with other operating systems having other communication endpoints other than sockets. It will be understood by those skilled in the art that various changes in form and detail may be made without departing from the spirit and scope of the invention.

I claim:

1. A system for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, wherein said first network domain has a network protocol architecture different from said second network domain, said system comprising:

at least one communication end point object in a layer of said first data processing system in said first network domain and at least one communication end point object in a layer of said second data processing system in said second network domain;

means, independently of an application running on either of said data processing systems, for automatically establishing, in said layer of said first data processing system and in said layer of said second data processing system, a connection between said first processing system and said second processing system and comprising means for mapping protocols between said first and second network domain; and

means for communicating over said connection between said first data processing system and said second data processing.

2. The system of claim 1 wherein the first network domain is a Transmission Control Protocol and the second network domain is a Systems Network Architecture.

3. A system for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, wherein said first network domain has a network protocol architecture different from said second network domain, said system comprising:

at least one communication end point object in a layer of said first data processing system;

an intermediate data processing system having at least one communication end point object in a layer of said intermediate data processing system;

at least one communication end point object in a layer of said second data processing system;

means, in said intermediate data processing system, for establishing automatically routed connections in said layer of said first data processing system, said layer of said second data processing system and said intermediate data processing system and comprising means for mapping protocols between said first and second network domain, said first and second processing systems each including means for executing respective application programs; and

means for communicating through said automatically routed connections between said first data processing system in said first network domain and said second data processing system in said second network domain.

4. The system of claim 3 wherein said means for communication immediately sends any data received from one end of said routed connection to said other end of said routed connection.

5. The system of claim 3 wherein said first data processing system includes a socket layer of socket code in said first data processing system;

said at least one communication end point object in a layer of said first data processing system is a socket in said socket layer of said first data processing system;

said intermediate data processing system includes a socket layer of socket code in said intermediate data processing system;

said at least one communication end point object in a layer of said intermediate data processing system is a socket in said socket layer of said intermediate data processing system;

said second data processing system includes a socket layer of socket code in said second data processing system;

said at least one communication end point object in a layer of said second data processing system is a socket in said socket layer of said second data processing system.

6. A system for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, wherein said first network domain has a network protocol architecture different from said second network domain, said system comprising:

at least one socket in a socket layer of said first data processing system in said first network domain;

at least one socket in a socket layer of said second data processing system in said second network domain;

means, independently of an application running on either of said data processing systems, for establishing in said socket layer of said first data processing system and in said socket layer of said second data processing system an automatically routed socket connection between said first data processing system and said second data processing system and

comprising means for mapping addresses between said first and second network domain; and

means for communicating through said socket connection between said first data processing system and said second data processing system.

7. A method for communicating between a first data processing system in a first network domain having a socket and a second data processing system in a second network domain, wherein said first network domain has a network protocol architecture different from said second network domain, said method comprising:

establishing, by said first data processing system, a socket in said second data processing system in said second network domain; and

invoking a routing facility to automatically establish a socket connection between said socket in said first data processing system and said socket in said second data processing system when said socket in said second data processing system is established and comprising means for mapping protocols between said first and second network domain;

communicating over said socket connection between said socket in said first data processing system in said first domain and said socket in said second data processing in said second domain; and

executing an application program on each of said first and second processing systems.

8. An operating system for use with a plurality of data processing systems for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, wherein said first network domain has a network protocol architecture different from said second network domain, said operating system comprising:

at least one socket in a socket layer of said first data processing system in said first network domain;

at least one socket in a socket layer of said second data processing system in said second network domain;

means, independently of an application running on either of said data processing systems, for automatically routing, in said socket layer of said first data processing system and in said socket layer of said second data processing system, a socket connection between said first data processing system and said second data processing system and comprising means for mapping addresses between said first and second network domain;

means for establishing said socket connection; and

means for communicating through said socket connection between said first data processing system and said second data processing system, wherein said data first and second processing systems each include means for executing respective application programs.

* * * * *

TAB 2

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

SN 304696

SERIAL NUMBER	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.
07/304,696	01/31/89	OWENS G	AT988089

MARILYN D. SMITH
IBM CORPORATION
11400 BURNET ROAD, 932/815
AUSTIN, TX 78758

EXAMINER	
COLEMAN,E	
ART UNIT	PAPER NUMBER
232	5

DATE MAILED: 03/29/90

This is a communication from the examiner in charge of your application.
COMMISSIONER OF PATENTS AND TRADEMARKS

☒ This application has been examined ☐ Responsive to communication filed on ______ ☐ This action is made final.

A shortened statutory period for response to this action is set to expire three (3) month(s), ______ days from the date of this letter.
Failure to respond within the period for response will cause the application to become abandoned. 35 U.S.C. 133

Part I THE FOLLOWING ATTACHMENT(S) ARE PART OF THIS ACTION:

1. ☒ Notice of References Cited by Examiner, PTO-892.
2. ☐ Notice re Patent Drawing, PTO-948.
3. ☒ Notice of Art Cited by Applicant, PTO-1449.
4. ☐ Notice of Informal Patent Application, Form PTO-152
5. ☐ Information on How to Effect Drawing Changes, PTO-1474.
6. ☐ ______

Part II SUMMARY OF ACTION

1. ☒ Claims 1-9 are pending in the application.

   Of the above, claims ______ are withdrawn from consideration.

2. ☐ Claims ______ have been cancelled.
3. ☐ Claims ______ are allowed.
4. ☒ Claims 1-9 are rejected.
5. ☐ Claims ______ are objected to.
6. ☐ Claims ______ are subject to restriction or election requirement.
7. ☒ This application has been filed with informal drawings under 37 C.F.R. 1.85 which are acceptable for examination purposes.
8. ☐ Formal drawings are required in response to this Office action.
9. ☐ The corrected or substitute drawings have been received on ______. Under 37 C.F.R. 1.84 these drawings are ☐ acceptable; ☐ not acceptable (see explanation or Notice re Patent Drawing, PTO-948).
10. ☐ The proposed additional or substitute sheet(s) of drawings, filed on ______, has (have) been ☐ approved by the examiner; ☐ disapproved by the examiner (see explanation).
11. ☐ The proposed drawing correction, filed ______, has been ☐ approved; ☐ disapproved (see explanation).
12. ☐ Acknowledgement is made of the claim for priority under U.S.C. 119. The certified copy has ☐ been received ☐ not been received ☐ been filed in parent application, serial no. ______; filed on ______.
13. ☐ Since this application apppears to be in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under Ex parte Quayle, 1935 C.D. 11; 453 O.G. 213.
14. ☒ Other – See attached

PTOL-326 (Rev.9-89)

EXAMINER'S ACTION

15. Claims 1-9 are presented for examination.

16. Claims 1-9 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

17. The scope of meaning of the following claim language is not clear:

a) " layer of said first data processing system" -- claim 1 (line 6),claim 3 (line 6),claim 7(line 13);

b) "layer of said second data processing system" -- claim 1 (line 7),claim 3 (line 8);

c) "said layer" -- claim 1 (line 10),claim 3,(lines 7,11);

d) "said routed connection" -- claim 1 (line 14), claim 3(line 15);

e)"socket in a socket layer" -- claim 5 (line 2), claim 9(line 7);

f) "socket layer" -- claim 6 (line 10), claim 9(line 12);

g)" said socket connection -- claim 6(line 14), claim 7(line 12), claim 9(line 16);

h) "creating...a socket" --claim 7 (line 5);

i) " connect a socket" -- claim 7 (line 8);

j) " said created socket" -- claim 7 (line 9), claim 8(line 13);

k) "said socket" -- cliam 7 (line 10);

l)" determining a means to make a connection" -- claim 8(line 5);

m)" creating a second socket" -- claim 8 (line 8);

n) "the determined connection" -- claim 8 (line 9);

o) " said determined connection" -- claim 8(line 11);

and

p) "said layer of said second...system" -- claim 9 (line 8).

18. As to 17(a,b) above,the meaning is ambiguous (i.e.,is there a physical layer or a layer of software or something else being claimed).

19. As to 17(c,d,j,n,o,p) above,the antecedent basis is not clear.

20. As to 17(e,f,g,h,i,k,m)above, the meaning is ambiguous (i.e., is there a physical socket or a software connection or something else being claimed).

21. As to 17(l) above,the meaning is ambiguous(i.e., it is unclear what criteria is used to make the claimed determination and it is unclear what determination is being claimed).

22. The following is a quotation of 35 U.S.C. 103 which forms the basis for all obviousness rejections set forth in this Office action:

> "A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made. Subject matter developed by another person, which qualifies as prior art only under subsection (f) and (g) of section 102 of this title, shall not preclude patentability under this section where the subject matter and the claimed invention were, at the time the invention was made, owned by the same person or subject to an obligation of assignment to the same person."

23. Claims 1-9 are rejected under 35 U.S.C. 103 as being unpatentable over Chang in view of Barzilai.

24. Chang taught (e.g., see figs 1-20) applicant's invention substantially as claimed including a data processing ("DP")system comprising :

a)communication end point object means and method (claims 1,6,7,8,9)(e.g., see col. 3, line 7);

b) independent automatic routing means and method(20)

(claims 1.6,7,8,9)(e.g., see col. 3. line 18,and col. 5, line 18);

c) means and method for communicating over routed connection means (claims 1,6,7,8,9) (e.g., see col.3, line 13).

25. Chang did not expressly detail(claims 1,6,7,8,9) that the communication end point means was in a layer of a first and a second data processing system . Chang , however, taught data exchange between remote programs (e.g., see col.3, line 18), and commands and routines that comprise a parameter that identifies the connection through a connection ID, a parameter which identifies a network function request and a parameter that points to a specific stucture in memory (e.g., see col.3, lines 45-53). Therefore it would have been obvious to a routineer in the DP art that Chang's DP system comprised a communication end point that was in some layer of the operating system software which was used in communication between DP systems.

26. As to the limitations of claim 2, Chang taught the use of Systems Network Architecture for communication (e.g., see col. 4, line 43), and accessing a network protocol (e.g., see col. 4, line 59).

27. Chang did not expressly detail (claim 3) an intemediate system. Barzilai, however, taught an intermediate data processing means (2) for routing communications between remote processing means(e.g., see fig. 8 and col. 6, line 34).

28. It would have been obvious to a routineer in the DP art to combine the teachings of Chang and Barzalai because they were both directed toward the problems of providing efficient program communication via to DP system networks.

29. As to the limitations of claims 4, Barzilai taught the data receiver sending an indication to slow down or stop the

sending of data when it could not handle the data at the current transmission rate(e.g.,see col. 3.,line 1) therefore it would have been obvious to a routineer that in the DP art that for the slow down or stop indication to be necessary in the remote communication via an intermediate processor means (2) then the intermediate processor would have had to immediately send the data to the receiving processor upon receipt of the data.

30. As to the "socket layer limitation of claims 5-9 it is unclear if the claimed invention operated exactly like Chang's DP system due to the discrepancy as detailed in paragraph 20 supra.

31. As to the operating system limitation of claim 8, Chang (e.g., see col.3, line 5) and Barzilai(e.g. see fig.8) further taught that their communication was via an operating system means.

32. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

Burke taught a DP system for handling unsolicited messages from lower tier controllers (e.g., see abstract).

Hart disclosed a DP system for bridging local area networks (e.g., see abstract).

Babecki disclosed a DP system for geographically distributed time-shared communication (e.g., see abstract).

Kepley disclosed a DP system with a message service system network (e.g., see abstract).

Dretzka disclosed a DP system using multiple physical links (e.g., see abstract).

Norstedt disclosed a DP system for network communication (e.g., see abstract).

Emerson disclosed a DP system with integrated message service (e.g., see abstract).

Yu disclosed a DP system for multiprocessor interrupt rerouting (e.g., see abstract).

33. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Eric Coleman whose telephone number is (703) -557 - 8014.

Any inquiry of a general nature or relating to the status of this application should be directed to the Group receptionist whose telephone number is (703) - 557 - 2878.

EC/ec

RAULFE B. ZACHE
PRIMARY EXAMINER
ART UNIT 232

TO SEPARATE, HOLD TOP AND BOTTOM EDGES, SNAP—APART AND DISCARD CARBON

8

FORM PTO-892 (REV. 3-78)

U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

NOTICE OF REFERENCES CITED

SERIAL NO. 304696

GROUP ART UNIT 232

ATTACHMENT TO PAPER NUMBER 5

APPLICANT(S) Gary L. Owens

U.S. PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
	A	4768150	8/88	Chang et al.	364	300	9/17/86
	B	4736369	4/88	Barzilai	370	94.1	6/13/86
	C	4855906	8/89	Burke	364	200	10/23/87
	D	4706081	11/87	Hart	370	61.	
	E	4500960	2/85	Babecki	364	200	
	F	4790003	12/88	Kepley	379	88	4/27/87
	G	4703475	10/87	Dretzka et al.	370	60	
	H	4586134	4/86	Norstedt	364	200	
	I	4612416	9/86	Emerson	379	88	
	J	4831518	5/89	Yu	364	200	8/26/86
	K						

FOREIGN PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	COUNTRY	NAME	CLASS	SUB-CLASS	PERTINENT SHTS. DWG	PP. SPEC.
	L								
	M								
	N								
	O								
	P								
	Q								

OTHER REFERENCES (Including Author, Title, Date, Pertinent Pages, Etc.)

R	
S	
T	
U	

EXAMINER Eric Cl

DATE 3/14/90

* A copy of this reference is not being furnished with this office action. (See Manual of Patent Examining Procedure, section 707.05 (a).)

TCORDEL0000089

TAB 3

MAIL ROOM
JUL 2 1990
PAT. & TRADEMARK OFF.

PATENT
07/304,696

#7/A

GROUP 230

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BH
7-9-90

In re application of	:	Before the Examiner:
G. L. Owens	:	E. Coleman
Serial No.: 07/304,696	:	Group Art Unit: 232
Filed: 1/31/89	:	Intellectual Property
Title: SYSTEM AND METHOD FOR	:	Law Department
INTERCONNECTING	:	International Business
APPLICATIONS ACROSS	:	Machines Corporation
DIFFERENT NETWORKS OF	:	11400 Burnet Road
DATA PROCESSING SYSTEMS	:	Austin, Texas 78758

9-2
7/12/90

June 29, 1990

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Box Non-Fee Amendments, Commissioner of Patents and Trademarks, Washington, D. C. 20231 on June 29, 1990.

Robert M. Carwell
Attorney for Applicants, Registration No. 28,499

Robert M. Carwell June 29 1990
Signature Date

AMENDMENT

Honorable Commissioner of Patents and Trademarks
Washington, D. C. 20231

Sir:

In response to the Office Action dated March 29, 1990, please amend the above identified Application as follows:

IN THE CLAIMS:

Cancel Claim 8 without prejudice to Applicant. Please amend Claims 1, 3-7 and 9 as follows:

PATENT
07/304,696

Claim 1 (Amended) A system for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, said system comprising:

at least one communication end point object in a layer of said first data processing system and at least one communication end point object in a said layer of [said] second data processing system;

means, independently of an application running on either of said data processing systems, for automatically [routing] establishing, in said layer of said first data processing system and in said layer of said second data processing system [having said at least one communication end point object], a connection between said first processing system and said second processing system; and

means for communicating over said [routed] connection between said first data processing system and said second data processing.

Claim 3 (Amended) A system for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, said system comprising:

at least one communication end point object in a layer of said first data processing system[,] ;

PATENT
07/304,696

an intermediate data processing system having at least one communication end point object in a layer of said intermediate data processing system;

at least one communication end point object in a layer of said second data processing system;

[in said layer of an intermediate data processing system, and in said layer of said second data processing system;]

means, in said intermediate data processing system, for establishing automatically routed connections in said layer of said first data processing system, said layer of said second data processing system, and said intermediate data processing system,

[automatically routing, in said layer having said at least one communication end point object, a connection] between said first processing system and said second processing system; and

means for communicating through said automatically routed connections [in said intermediate processing system] between said first data processing system and said second data processing system.

Claim 4 (Amended) The system of claim 3 wherein said means for communication [in said intermediate processing system] immediately sends any data received from one end of said routed connection to said other end of said routed connection.

PATENT
07/304,696

Claim 5 (Amended) The system of claim 3 wherein said first data processing system includes a socket layer of socket code in said first data processing system;

said at least one communication end point object in a layer of said first data processing system is a socket in said socket layer of said first data processing system; [said at least one communication end point object is a socket in a socket layer of each of said data processing systems.]

said intermediate data processing system includes a socket layer of socket code in said intermediate data processing system;

said at least one communication end point object in a layer of said intermediate data processing system is a socket in said socket layer of said intermediate data processing system;

said second data processing system includes a socket layer of socket code in said second data processing system;

said at least one communication end point object in a layer of said second data processing system is a socket in said socket layer of said second data processing system.

Sub C3

Claim 6 (Amended) A system for communicating between a first data processing system in a first network domain

PATENT
07/304,696

and a second data processing system in a second network domain, said system comprising:

at least one socket in a socket layer of said first data processing system [and in said layer of said second data processing system];

at least one socket in a socket layer of said second data processing system;

a2

means, independently of an application running on either of said data processing systems, for establishing in said socket layer of said first data processing system and in said socket layer of said second data processing system an automatically routed socket [automatically routing, in said socket layer, a] connection between said first data processing system and said second data processing system; and

means for communicating through said socket connection between said first data processing system and said second data processing system.

Sub B2

Claim 7 (Amended) A method for communicating between a first data processing system in a first network domain having a socket and a second data processing system in a second network domain, said method comprising:

B3

establishing [creating], by said first data processing system, a socket in said second data processing system in said second network domain; and

PATENT
07/304,696

invoking a routing facility to automatically establish a socket connection between said [connect a] socket in said first data processing system [to said created] and said socket in said second data processing system when said socket [is created] in said second data processing system is established [network domain]; and

communicating over said socket connection between said socket in said first data processing system in said first domain and said [created] socket in said second data processing system in said second domain.

[Cancel Claim 8.]

Sub C4

Claim 9 (Amended) An operating system for use with a plurality of data processing systems for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, said operating system comprising:

at least one socket in a socket layer of said first data processing system [and in said layer of said second data processing system];

at least one socket in a socket layer of said second data processing system;

means, independently of an application running on either of said data processing systems, for automatically routing, in said socket layer of said first data processing system and in said socket layer of said second data processing system, a socket connection between

said first data processing system and said second data processing system; [and]

means for establishing said socket connection; and

means for communicating through said socket connection between said first data processing system and said second data processing system.

Q2 B

REMARKS

Claims 1, 3-7 and 9 have been amended herewith and Claim 8 cancelled without prejudice to Applicant.

In paragraph 16 & 17 of the subject Office Action, the Examiner has rejected Claims 1-9 under 35 U.S.C. 112, second paragraph as being indefinite. The Examiner stated that the scope of meaning of various claim language cited in paragraph 17 is not clear. The Examiner has further noted with respect to paragraph 19 of the Office Action that antecedent basis is not clear with respect to various cited claim language, and in paragraph 21 that as to Claim 8, line 9, the meaning is ambiguous.

With respect to the foregoing, Applicant believes that with the claim language amendments herewith submitted, the grounds for rejecting the claims for the aforesaid reasons stated by the Examiner are no longer present. And accordingly Applicant respectfully requests the Examiner to withdraw rejection of these claims.

In paragraph 18 and 20, with respect to pending claims cited therein, the Examiner has stated that claim language is ambiguous, inquiring whether the layers and sockets recited in the claims are physical or software implemented. The

layers and sockets in the context of Applicant's Specification may be clearly interpreted by one of ordinary skill in the art as being software implemented, support for which may be seen in the excerpt of the "Dictionary of Computing", 8th edition (March, 1987), copyright 1987, IBM, which has been submitted herein. The Examiner is respectfully requested to withdraw rejection of the claims under 35 U.S.C. 112 on the grounds of ambiguity.

Claims 1-9 were rejected under 35 U.S.C. 103 as being unpatentable over Chang in view of Barzilai.

The Examiner is requested to note that Applicant has invented a system and method for automatically routing a connection between data processing systems in different network domains. In this manner, an application running on a data processing system utilizing a network domain such as TCP can automatically make a connection to another data processing system utilizing a different network domain such as SNA. The term "domain", with reference to Applicant's specification, has a precise meaning as set forth therein on page 4, lines 20-28.

The Examiner is respectfully requested to note that with respect to every independent claim now pending, there is a limitation not only in the preamble but in the body of the claims themselves which call for first and second data processing systems in respective different first and second network domains. The heart of the invention relates to the routing of data between data processing systems operating in these differing respective network domains.

Referring first to the Chang '150 reference cited by the Examiner, the Examiner is first requested to note that Chang is dealing with a single network domain, e.g. SNA, wherein an SNA session defines the network path that links two logical

units which are communicating (column 2, line 21-29). More particularly, Chang is dealing with the problem of handling a change in network protocols. He observed at column 2, line 52, as did Applicant in his Specification on page 2, line 18, that typically the application program must be rewritten to adjust the changes. Chang went further to note that a fundamental deficiency in this approach is lack of an interface to an application program for accessing networking functions so as to facilitate such adaptation to changing protocols. Chang thus addressed a problem entirely different from that of Applicant by providing an interface externalizing the operating system commands, thereby giving an application program direct interaction with a network environment while obviating the need for intimate knowledge of the network protocols. An application transaction program was thereby allowed to access the SNA architecture through the application's use of the operating system calls. Whereas Chang observed that his application program interface 20 was applicable to other systems, the application was to other systems accessing the identical SNA protocol (column 4, lines 54 et seq). Accordingly, Chang in no way taught or even remotely suggested Applicant's provision for a system for communicating between multiple data processing systems operating in respective different network domains as, for example, more particularly recited with respect to dependent Claim 2, for example, wherein the first network domain is a TCP and the second network domain is an SNA. The Examiner has stated, in correlating the Chang reference to Applicant's disclosure, that Chang discloses an independent automatic routing means. Even assuming, arguendo, that this is true, Applicant is claiming a system and method for automatically establishing a connection between data processing systems

each in respective different network domains, thereby patentably distinguishing over Chang.

The Examiner has cited Barzilai, '369, as teaching intermediate data processing means in the manner of Applicant for routing communications between remote processing means and, more specifically, with reference to Applicant's Claim 4, that Barzilai has disclosed an intermediate processor acting immediately to send data to a receiving processor.

The Examiner is requested to note that, as with Chang, the main concern in Barzilai related to data processing system transmission in a single network domain, e.g. SNA (column 1, line 57-60). More particularly, whereas Chang was concerned with an application program interface (API) to this <u>single</u> SNA protocol, unlike Applicant's as hereinbefore noted, Barzilai was focusing on the problem of transmission across this <u>single</u> SNA network. Barzilai observed that static or fixed session-level pacing between logical units (LU) in a <u>single</u> structured architecture such as SNA, was a problem in the prior art in controlling flow of information in a communications network. This was due to the fact that packets in a window were fixed at session activation and could not be adjusted dynamically. Thus his contribution was to provide an adaptive session pacing mechanism (column 2, lines 20-30 and lines 51-53), a problem totally foreign to that being addressed by Applicant. It is clear that Barzilai, while appreciating adaptation of his invention to an architecture other than SNA, was not contemplating a system for communication between data processing systems in separate and distinct network domains as clearly recited in Applicant's claims. This may be seen with reference to column 4, lines 34-38 of Barzilai. Moreover, particularly in view of the fact that Chang and Barzilai were not only

addressing different problems but also problems completely different from Applicant, the Examiner has cited no reason, as is required, why the teachings of the references would be obvious to combine. Neither reference either alone or combination, teaches or remotely suggests a system for communicating between multiple data processing systems operating in separate and distinct network domains nor a system and method for establishing automatically routed communication between such systems as clearly and distinctly claimed by Applicant. Still further, even assuming for the moment that Barzilai teaches a means for communication in an intermediate processing system, as Claim 4 depends from Claim 3 wherein the communicating means includes the limitation of communication between first and second data processing systems each having their own network domains, it is submitted that this patentably distinguishes over the references of record.

In paragraph 30 of the instant Office Action, the Examiner has stated that it is unclear whether the socket layer limitation of Claims 5-9 distinguish over Chang inasmuch as it is unclear whether the recited socket is physical or software-implemented. The Examiner will note, now that the Applicant has cleared up the discrepancy, that use of the term socket has a precise meaning set forth in Applicant's Specification, page 3, line 26, through page 4, lines 1-9. The sockets called out in Applicant's claims are implemented in respective different network domains and accordingly patentably distinguish over Chang which neither teaches nor even remotely suggests Applicant's sockets, let alone provision for automatically establishing or routing communications between these sockets corresponding to different network domains.

For all the foregoing reasons, the Examiner is respectfully requested to withdraw rejection of Claims 1-7 & 9 under 35 U.S.C. 103, to allow such claims and to pass this case to issue. If the Examiner feels that a telephone conference with Applicant's attorney would expedite prosecution, he is requested to contact the attorney at the hereinbelow telephone number.

Respectfully submitted,

G. L. OWENS

BY Robert M. Carwell

Robert M. Carwell
Attorney for Applicants
Registration No. 28,499
(512) 823-1017

RMC/bas

MAIL ROOM JUL 2 1990 PAT. & TRADEMARK OFF.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

P 23

Case Docket No. AT9-88-089
PATENT

In re application of: G. L. Owens

Serial No.: 07/304,696

Filed: 1/31/89

For: SYSTEM AND METHOD FOR INTERCONNECTING APPLICATIONS ACROSS DIFFERENT NETWORKS OF DATA PROCESSING SYSTEMS

RECEIVED
JUL 05 1990
GROUP 230

THE COMMISSIONER OF PATENTS & TRADEMARKS
Washington, D.C. 20231

Sir:

Transmitted herewith is an Amendment in the above-identified Application.

[X] No additional fee is required.

[ ] The fee has been calculated as shown below:

	(Col. 1) CLAIMS REMAINING AFTER AMENDMENT		(Col. 2) HIGHEST NO. PREVIOUSLY PAID FOR	(Col. 3) PRESENT EXTRA
TOTAL	* 8	MINUS	** 20	= 0
INDEP.	* 5	MINUS	*** 6	= 0
[ ] 1ST PRESENTATION OF MULTIPLE DEP. CLAIM				

OTHER THAN A SMALL ENTITY RATE	ADDIT. FEE
x 12=	$ 0
x 36=	$ 0
+120=	$ 0
TOTAL	$ 0

* If the entry in col. 1 is less than the entry in col. 2, write "0" in col. 3.
** If the "Highest No. Previously Paid For" IN THIS SPACE is less than 20, write "20" in this space.
*** If the "Highest No. Previously Paid For" IN THIS SPACE is less than 3, write "3" in this space.
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found from the equivalent box in col. 1 of a prior amendment or the number of claims originally filed.

[ ] Please charge my Deposit Account No. 09-0451 in the amount of $________. A duplicate copy of this sheet is enclosed.

[X] The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 09-0451. A duplicate copy of this sheet is enclosed.

[X] Any additional fees required under 37 CFR §1.16 for the presentation of extra claims.

[X] Any patent application processing fees under 37 CFR §1.17.

Respectfully submitted,

By Robert M. Carwell

Robert M. Carwell
Registration No. 28,499
Intellectual Property Law Dept.
IBM Corporation
11400 Burnet Road - 4054
Austin, Texas 78758 (512) 823-1017

TAB 4

#19 Bd attachment D

Response Under 37 CFR 1.116
Expedited Procedure
Examining Group 232

PATENT

RECEIVED
FEB 24 1992
GROUP 230

MAIL ROOM
FEB 18 1992
PAT. & TRADEMARK OFFICE

BOX AF

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of : Before the Examiner:
G. L. Owens : E. Coleman
Serial No.: 07/304,696 : Group Art Unit: 232
Filed: 1/31/89 : Intellectual Property
Title: SYSTEM AND METHOD FOR INTERCONNECTING APPLICATIONS ACROSS DIFFERENT NETWORKS OF DATA PROCESSING SYSTEMS : Law Department, International Business Machines Corporation, 11400 Burnet Road, Austin, Texas 78758

DG 2/25/92

February 14, 1992

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Box AF, Commissioner of Patents and Trademarks, Washington, D. C. 20231 on February 14, 1991.

Wayne P. Bailey
Attorney for Applicants, Registration No. 34,289

W Bailey 2/14/92
Signature Date

Please Enter EG 3/3/92

AMENDMENT AFTER FINAL PURSUANT TO 37 CFR 1.116

Honorable Commissioner of Patents and Trademarks
Washington, D. C. 20231

Sir:

In response to the Office Action dated 11/5/91, please amend the above identified Application as follows:

PATENT
07/304,696

IN THE CLAIMS:

Claim 1 (Thrice Amended) A system for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, wherein said first network domain has a network protocol architecture different from said second network domain, said system comprising:

at least one communication end point object in a layer of said first data processing system in said first network domain and at least one communication end point object in a layer of said second data processing system in said second network domain;

D1

means, independently of an application running on either of said data processing systems, for automatically establishing, in said layer of said first data processing system and in said layer of said second data processing system, a connection between said first processing system and said second processing system and comprising means for mapping protocols between said first and second network domain; and

means for communicating over said connection between said first data processing system and said second data processing.

Claim 3 (Four Times Amended). A system for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, wherein said first network domain

D2

has a network protocol architecture different from said second network domain, said system comprising:

at least one communication end point object in a layer of said first data processing system;

an intermediate data processing system having at least one communication end point object in a layer of said intermediate data processing system;

D2

at least one communication end point object in a layer of said second data processing system;

means, in said intermediate data processing system, for establishing automatically routed connections in said layer of said first data processing system, said layer of said second data processing system and said intermediate data processing system and comprising means for mapping protocols between said first and second network domain, said first and second processing systems each including means for executing respective application programs; and

means for communicating through said automatically routed connections between said first data processing system in said first network domain and said second data processing system in said second network domain.

D3

Claim 6 (Thrice Amended). A system for communicating between a first data processing system in a first network domain and a second data processing system in a

second network domain, wherein said first network domain has a network protocol architecture different from said second network domain, said system comprising:

at least one socket in a socket layer of said first data processing system in said first network domain;

at least one socket in a socket layer of said second data processing system in said second network domain;

means, independently of an application running on either of said data processing systems, for establishing in said socket layer of said first data processing system and in said socket layer of said second data processing system an automatically routed socket connection between said first data processing system and said second data processing system and comprising means for mapping addresses between said first and second network domain; and

D3

means for communicating through said socket connection between said first data processing system and said second data processing system.

Claim 7 (Thrice Amended). A method for communicating between a first data processing system in a first network domain having a socket and a second data processing system in a second network domain, wherein said first network domain has a network protocol architecture different from said second network domain, said method comprising:

PATENT
07/304,696

establishing, by said first data processing system, a socket in said second data processing system in said second network domain; and

invoking a routing facility to automatically establish a socket connection between said socket in said first data processing system and said socket in said second data processing system when said socket in said second data processing system is established and comprising means for mapping protocols between said first and second network domain;

DB

communicating over said socket connection between said socket in said first data processing system in said first domain and said socket in said second data processing in said second domain; and

executing an application program on each of said first and second processing systems.

Claim ~~9~~ 8 (Four Times Amended). An operating system for use with a plurality of data processing systems for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, wherein said first network domain has a network protocol architecture different from said second network domain, said operating system comprising:

DH

at least one socket in a socket layer of said first data processing system in said first network domain;

PATENT
07/304,696

at least one socket in a socket layer of said second data processing system in said second network domain;

means, independently of an application running on either of said data processing systems, for automatically routing, in said socket layer of said first data processing system and in said socket layer of said second data processing system, a socket connection between said first data processing system and said second data processing system <u>and comprising means for mapping addresses between said first and second network domain</u>;

D4

means for establishing said socket connection; and

means for communicating through said socket connection between said first data processing system and said second data processing system, wherein said data first and second processing systems each include means for executing respective application programs.

<u>Remarks</u>

Applicants wish to thank the Examiner for the telephone interview on January 29, 1992. Although no agreement was reached, Applicants now better understand the Examiner's position and present this amendment in response thereto.

Claim 1-7 and 9 are pending in this application, and stand finally rejected pursuant to the Examiner's Office Action dated 11/5/91. The Examiner is requested to reconsider, and withdraw, the final rejection of these Claims 1-7 and 9 based on the following comments.

Claims 1, 3-7 and 9 were rejected by the Examiner under 35 U.S.C. 102 as being anticipated by Bishop ('877). The Examiner states that Bishop shows communication endpoints in a layers of first and second DP systems, automatic routing means, means for communicating over the routed connection, an intermediate DP system with an endpoint in a layer, means for establishing a socket means in a second DP system, and means and method for executing an application program in first and second processing systems.

In the most recent Office Action, the Examiner further states in paragraph 23 that Bishop teaches communication of messages across domains in a network environment of computers where each domain comprises a differing DP system processing different processes. Applicants traverse this assertion as follows.

Bishop fails to disclose communications between *differing network domains*. As defined in Applicants' Specification, page 1, lines 26-30, network domains mean network protocol architectures. Bishop teaches a *single* application program running by using a plurality of processors. Applicants, on the other hand, are claiming a means and method for allowing *multiple* application programs to communicate with each other over a *cross-domain* network configuration. This is not taught by Bishop, and thus the rejected claims based on this asserted teaching of Bishop was erroneous. Claims 1, 3-7 and 9 include the limitation of connecting data processing systems in differing network domains. This limitation is not described or suggested by Bishop. Rather, Bishop only supports a single domain connection between multiple processors. Differing network protocol architectures are not taught or supported by Bishop.

After discussion with the Examiner in a telephonic interview, the Examiner stated that the term 'domain' was being given its broad meaning as defined in the dictionary, and not the more limited definition as defined in Applicants' specification. Claims 1, 3-7 and 9 have been amended to clearly show that 'domain' is intended to mean network protocol architecture, as defined in the Specification. This amendment does not result in any new search being required, as the limitations of Claim 2 (which has been previously searched by the Examiner) explicitly recite this limitation for particular protocols. Further, this amendment was not previously possible as Applicants are allowed to be their own lexicographer, and the limitation included in the amended claim preamble had previously been defined in the Specification, and subsequent amendments expressly stated that this definition was what was intended by Applicants when interpreting the claims. Thus, the Examiner is requested to enter this amendment to the claims. Applicants have further amended Claims 1, 3-7 and 9 to breath life and meaning to this preamble modification, and these amended claims clearly recite the requisite mapping which is done for dissimilar domains to achieve the claimed cross domain connection. (Specification support at page 18, line 22 through page 22, line 34).

Further, it would not be obvious to modify Bishop to support cross domain connections, as the entire Bishop scheme is based on process tables with appropriate pointer entries (see Bishop Fig. 4) which are similarly used in nodes being interconnected via virtual channel (Bishop, Col. 9, lines 2-10 and Col. 10, lines 16-35). The duplication of some entries and not others between processes allows for kernel determination of whether it is executing a stub or user

process (Bishop, Col. 10, lines 31-35). If differing domains were being interconnected in Bishop, these tables would have to be specific to the particular domain being interconnected and the Bishop process determination methods would thus become inoperable. This resultant Bishop system would then fail, as no mechanism would exist for a kernel to know how it fit into the overall system and workload parsing. This failure of system operability strongly evidences non-obviousness..

The Examiner also states in paragraph 22 that Bishop teaches that stub processes route packets between processes using an intermediate DP system; and that Bishop teaches means for executing plural programs. To substantiate this statement, the Examiner points to Bishop Col. 5, lines 38-47). This passage fails to disclose the above listed limitations of independent Claim 3, however. Rather, the Bishop passage discusses sending status messages to multiple processors when a process is to be moved from one processor to another processor. It does not disclose 'means for establishing automatically routed connections', as claimed by Applicants. The connections in Bishop are fixed, with no teaching of automatically routing *connections*.

In the telephonic interview, the Examiner reiterated that the cited passage was being read to include rerouting by an intermediate processor, because of the Bishop statement that

> "The kernel of other computers will now direct any signals for olduser process 109 to newuser process 111."

The Examiner's position appears to be that signals could be interpreted to include communication messages generated from

yet a third computer. Thus, this third computer would generate a communication signal to the kernel (in a second computer), and this kernel would direct the signal to newuser process (in a first computer). Applicants stated that the kernel is only receiving signals which were generated for processes *internal* to the particular computer which the kernel was executing upon, and thus no intermediate routing was occurring, as the Bishop kernel is only directing internally generated signals. To substantiate this assertion, Applicants have attached, in Attachment A, a section from the book entitled 'UNIX Time-Sharing System: UNIX Programmer's Manual', by Holt, Rinehart and Winston of Bell Telephone Laboratories, 1983. Applicants have also attached, in Attachment B, Chapter 8 of a book entitled 'The UNIX Operating System Book" by Banahan and Rutter, 1983. These articles will now be discussed in detail to show that the use of the term 'signal' by Bishop has a very specific meaning in Bishop's UNIX operating environment context, and that the interpretation being given by the Examiner is inconsistent with this meaning.

Referring specifically to Attachment A, it is seen that UNIX signals are generated by some abnormal event, and that normally all signals cause termination of the receiving process. This is inconsistent with the interpretation given by the Examiner, where a signal from one processor is received by a kernel in another processor and rerouted to yet a third processor. This is so for two reasons. First, interprocessor communication alleged by the Examiner could not be construed to be an abnormal event. Secondly, interprocessor communications would fail if the signal caused the receiving process to terminate. Notwithstanding this clear distinction, Applicants wish to further direct the

Examiner's attention to Bishop, Col. 5, lines 48-53. Here is described that the newuser process must *itself* retrieve any signals which were received by olduser process prior to other computers being notified that the extended process had migrated to computer 104 of Fig. 3. This is further proof that automatic intermediate processor rerouting is not occurring in Bishop, as the receiving processor must manually retrieve messages sent to the alleged intermediate processor. Once the sending processors have been notified of the change in user process location, the sending processors route messages directly to the new user process.

Further, Attachment B, describes the interaction between processes and the kernel in a UNIX environment. Besides containing a general description of how processes interact with a system kernel, pages 116-117 further substantiates that signals are abnormal conditions in UNIX which generally result in process termination. The description of process/kernel interaction is also enlightening in understanding the teachings of Bishop.

Bishop teaches the ability to extend a process over a number of computers (Col. 4, lines 17-18 and Figure 1). The extended process is a collection of individual special processes running on separate computers. These special processes are also referred to as the primary/user process and auxiliary/stub processes. In referring to Figure 3 of Bishop, the primary/user process is located in computer 104, with auxiliary/stub processes existing in computers 101 and 103. Bishop explicitly states that stub processes do not communicate with each other, and that *all communication* from other processes within the system illustrated in Figure 1 are directed to the user process (Bishop, Col. 8, lines 65-68). The user process is in control of the system, and sends

various messages to the auxiliary processes. There is no teaching or suggestion of interprocess communication between auxiliary processes. Further, there is no teaching or suggestion of automatically routing connections in an intermediate data processing system.

Finally, it should be noted that the Bishop system merely provides for a *single process* to be extended across a plurality of computers (Bishop, Col. 4 lines 31-42 and 51-55). Applicants are claiming in Claims 3-5 and 9 that respective application programs are running in the first and second data processing systems. This is not taught by Bishop.

As all the claimed limitations of amended Claims 1, 3, 6, 9, and dependants thereof, are not taught or suggested by Bishop, the rejection of Claims 1, 3-7 and 9 should be withdrawn.

The Examiner next rejected Claim 2 under 35 U.S.C. 103 as being unpatentable over Bishop in view of McKay. The Examiner relies on the earlier described grounds for rejecting Claims 1, 3-7 and 9, and further adds that McKay taught the use of network protocols SNA and FEP in the automatic establishment of connections between DP system as shown at col. 10, line 29. Applicants traverse this rejection as follows.

First, as Claim 2 contains all the limitations of Claim 1, which has been shown above to be patentable, this dependent Claim 2 is similarly patentable in view of the combination of references.

Further, the McKay reference was improperly combined with the Bishop reference. In establishing a prima facie case of obviousness, the Examiner must present a rationale or reason why the artisan would have been led to do that which

the claims specify as the invention. The reason must stem from something suggested by the prior art or from demonstrated common knowledge of the artisan, or both, and not from the inventor's disclosure. See Uniroyal, Inc. v. Rudkin-Wiley Corp., 837 F.2d 1044, 5 USPQ2d 1434 (Fed. Cir. 1988); Ashland Oil, Inc. v. Delta Resins & Refactories, Inc., 776 F.2d 281, 227 USPQ 657 (Fed. Cir. 1985) and In re Sernaker, 702 F.2d 989, 217 USPQ 1 (Fed. Cir. 1983).

It is insufficient that the prior art disclosed the components of the patented device, either separately or used in other combinations; there must be some teaching, suggestion, or incentive to make the combination made by the inventor, Interconnect Planning Corp. v. Feil, 227 USPQ 543, 551 (Fed. Cir. 1985). The mere fact that the prior art could be so modified would not have made the the modification obvious unless the prior art suggested the desirability of the modification, In re Laskowski, 10 USPQ 2d 1397 (Fed. Cir. 1989).

The motivation asserted by the Examiner to combine such references is that it would be obvious to a person of ordinary skill in the art to combine these references as they were both directed toward a common problem of providing efficient automatic establishment of connections between DP system for providing efficient communication therebetween. The mere fact that a references are directed to solving a general problem of DP communications is insufficient, in and of itself, to justify a proper combination of references.

It should also be noted that the two systems are architecturally distinct, having differing requirements, implementations, and system tradeoffs. A person of ordinary skill in the art could not merely dissect a single function from one architecture and insert it into a dissimilar architecture.

Claim 2 has thus been improperly rejected under 35 U.S.C. 103. Specifically, the Bishop teaching requires that the same operating system be running on each interconnected computer. The ability to extend a Bishop process across multiple computers requires this, based on the detailed Bishop implementation of replicating operating system control blocks across each computer. The Bishop system is tightly-coupled, using a common bus for computer interconnection (see Bishop Fig. 1). The McKay system does not teach how to modify multiple, tightly coupled computers. McKay's design is a loosely-coupled computer environment which allows dissimilar terminals to attach to a host via an incompatible intermediate network. The terminals do not have an operating system, much less one that is identical to the host computer which has been modified as taught by Bishop. A person of ordinary skill in the art would not be motivated to combine two such dissimilar systems.

Based on the above, the Examiner is requested to reconsider the final rejection of the Claims 1-7 and 9, withdraw same, and pass these claims to issue as all basis for rejection have been successfully traversed.

Applicants further request that the Examiner acknowledge the review of prior art submitted by a Applicants in a Supplemental IDS dated August 5, 1991, as no document has been received by Applicants indicating such review.

Respectfully submitted,

G. L. OWENS

BY W Bailey

Wayne P. Bailey
Attorney for Applicants
Registration No. 34,289
(512) 823-1012

TAB 5

07 304696

MB162117956

MAIL ROOM
JAN 31 1989
47
PAT. & TRADEMARK OFF.

APPLICATION
FOR
UNITED STATES LETTERS PATENT

INTERNATIONAL BUSINESS MACHINES CORPORATION

AT9-88-089

SN07/304,696

Abstract of the Disclosure

The system and method of this invention automatically routes a connection between data processing systems in different network domains. As an example, an application running on a data processing system utilizing a network domain such as TCP (Transmission Control Protocol), can automatically make a connection to another data processing system utilizing a different network domain such as SNA (Systems Network Architecture). The connection is automatically performed in the layer containing the communication end point objects. In a preferred embodiment, the connection is automatically performed in the socket layer of the AIX operating system, or in the socket layer of other operating systems based upon the Berkeley version of the UNIX operating system.

1

AT9-88-089 7

07 304696 A MB162-17956

MAIL ROOM JAN 31 1989 PATENT & TRADEMARK OFF.

NP

NK

Ins E1

SU P

1

~~Description~~

A SYSTEM ~~AND METHOD~~ FOR INTERCONNECTING APPLICATIONS ACROSS DIFFERENT NETWORKS OF DATA PROCESSING SYSTEMS

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

CLU|G

Background of the Invention

PB 1G P

Field of the Invention

This invention relates to a network of data processing systems, and more specifically to the interconnection of a plurality of data processing systems between different network protocol domains, such as the different network protocol domains of SNA and TCP/IP.

PB 2G P

Description of the Related Art

A system having multiple domains has at least one data processing system that is interconnected to at least two different data processing systems through at least two different network domains, i.e. network protocol architectures. A problem with multiple domains is the difficulty in allowing communication between machines which are connected to another type of network. For example, a data processing system utilizing SNA LU 6.2 as its network protocol can not automatically communicate with another data processing

B

2

AT9-88-089

system utilizing TCP/IP as its network protocol. Both
A
SNA LU 6.2 and TCP/IP are examples of stream protocols where data flows as a stream of indeterminate lengths, and the bytes are delivered in the correct order. The problem is routing a stream of bytes from a data processing system that utilizes a reasonably equivalent protocol, such as a stream protocol, to another data processing system that also utilizes a reasonable equivalent protocol, such as the stream protocol of this example, but wherein the two protocols are not
B
the exact same protocol, such as SNA LU 6.2 and TCP/IP.

It is known to solve the above problem at the application program level. An application program which is running on a data processing system at one end of the connection may be designed to utilize a specific network protocol. In this case, it is known to modify the application in order to reimplement the application to work over another protocol. This requires changing the source program code of the original application by some amount. Depending upon how the application program was originally designed, this may require a substantial amount of changes to the program code.

It is also known to solve the above problem by implementing the same protocol on both machines. For example, in order to use an SNA transaction application running in an SNA network, to apply transactions against data processing systems utilizing a TCP network, one could reimplement that transaction application against TCP by then putting TCP on the client data processing system, put IP over SNA, and gateway between the two. The client data processing system can then be implemented utilizing TCP/IP. The problem with this approach is having to reimplement

3

the application to utilize the different protocol at one end of the network or the other. This is especially burdensome if the application is large and complex.

There are some application level protocols that handshake back and forth over SNA, e.g. 3270 SNA. These have their own data format with meta-data in the data stream. There are other application level protocols, such as Telnet over TCP, that talk back and forth that have meta-data and data in the data stream. However, one can not get these two to talk together since these two have different data and meta-data in their data streams.

If an application utilized one protocol, and that application were to run on a data processing having a different protocol, knowing the data stream format, one could write the client half of the application on the data processing system utilizing the other protocol.

Therefore, in order to extend network connectivity, it is known to reimplement the application to utilize the different protocol, put one protocol on top of the other, and gateway between the two. It is also known to build a larger network utilizing each type of protocol through replication and duplication.

The term "sockets" is an application program interface (API) that was developed for the Berkeley version of AT&T's UNIX[1] operating system for interconnecting applications running on data processing systems in a network. The term socket is used to

B P H 3

NP NK

---

[1]UNIX is licensed and developed by AT&T. UNIX is a registered trademark of AT&T in the U.S.A. and other countries.

define an object that identifies a communication end point in a network. A socket can be connected to other sockets. Data can go into a socket via the underlying protocol of the socket, and be directed to appear at another socket. A socket hides the protocol of the network architecture beneath a lower layer. This lower layer may be a stream connection model (virtual circuit), or a datagram model (packet), or another model.

XI

P A stream connection model refers to a data transmission in which the bytes of data are not separated by any record or marker. A virtual circuit implies that there appears to be one communications end point connected to one other communications endpoint. When the connection is established, only those two end points can communicate with each other.

Sockets are typed by domain (address family or network type), and model type (stream, datagram, etc.). If needed, the socket can be further specified by protocol type or subtype. The domain specifies the addressing concept utilized. For example, there is an internet IP domain, and also a SNA domain for networks utilizing TCP and SNA, respectively. As used herein, the word "domain" is used to refer to the address family of a socket, and not to a domain-naming domain. A domain-naming domain is a concept of a related group of hierarchical addresses, wherein each part of the address is separated by a delimiter, such as a period.

Since a socket is specified by the domain, sockets do not allow cross domain connections. This means that if an application program creates a socket in the Internet (Darpa) domain, it can only connect to sockets in that same domain. Note: "Darpa" is used to specify that Internet, short for internetworking, is not only used herein both to generically specify the

internet layer of a particular protocol family which contains means for forwarding, routing control, and congestion control, etc., but also as a name for a particular implementation of an internet called the Internet or the Darpa Internet, or the Arpa Internet. Another name for this internet layer is the Internet Protocol (IP). TCP/IP is also commonly used to refer to this protocol.

Originally, the requirement that a socket can only connect to sockets in the same domain was a reasonable restriction. This simplified the program code when there was only one really useful domain anyway. With the advent of the usage of other domains (specifically SNA), cross domain connections have become desirable. For example, cross domain connections would allow mailers to transport mail among domains. Also, cross domain connections would allow programs to communicate using the existing communication networks.

CL-U/C
P

Summary of the Invention

It is therefore an object of this invention to automatically route connections between data processing systems that utilize different protocols, independently of said applications running on said data processing systems.

It is a further object of this invention to route, at the socket level, between two networks when a cross-domain connection attempt is detected.

It is a further object of this invention to facilitate the interconnection between data processing systems by allowing socket based applications to easily span across different networks.

6

It is a further object of this invention to communicate between data processing systems in which one of the data processing systems utilizes TCP/IP and the other data processing system utilizes SNA.

It is a further object of this invention to communicate between two data processing systems via a third data processing system utilized as a TCP to SNA gateway.

It is a further object of this invention to communicate through a connection between two data processing systems both utilizing TCP on each of their local Internets, by bridging the network connection with a long haul SNA connection.

The system and method of this invention automatically routes a connection between data processing systems, independently of an application running on the data processing systems, having different network domains. The preferred embodiment describes the cross domain interconnections with reference to the different network domains of TCP (transmission control protocol) and SNA (systems network architecture).

The routing is automatically performed at a layer which contains the communication end point objects. In the AIX[2] operating system, and other operating systems based upon the Berkeley version of the UNIX operating system, this layer is called the socket layer.

P

An intermediate processing system is utilized to gateway between a processing system utilizing a network domain such as TCP, and another processing system utilizing a different network domain such as

---

[2]Trademark of IBM Corporation

7

AT9-88-089

SNA. Alternatively, the client data processing system can be implemented utilizing TCP/IP which can then be gatewayed through socket routing on the same machine into an SNA data stream without an intermediate processing system performing the socket routing.

P

In any event, the socket layer which performs the socket routing contains facilities to automatically route a connection across different domains.

In the client processing system which is attempting to create a connection, a socket is created in a particular domain. If the socket is in a different domain, the socket does not fail if the socket routing facility of this invention is implemented. The **connect** function is modified to catch the attempts at a cross domain connection. If a **connect** function is attempted on a socket in a different domain, then the socket routing facility of this invention is invoked.

Alternatively, a **connectto** function can be implemented which takes the place of and combines the functions of the **socket** function and the **connect** function. With the **connectto** function, a socket is not created until the route is known. This alleviates the unnecessary work of creating a socket which may fail, and then performing actions as a result of the failed socket. The **connectto** function determines how a connection can be made, and then creates a socket in the domain that is needed to establish the determined connection.

Through either of the above approaches, a connection to a socket in a different domain can be made through an intermediate socket. When data arrives from one end of the connection to the intermediate socket, the intermediate socket immediately sends the data to the other end of the connection instead of

9

queuing the data for process intervention at the intermediate processing system.

In addition, if the intermediate socket is queried for the address of the other end of the connection, the intermediate socket identifies the connecting host as opposed to the intermediate host. In this way, the socket routing facility of the intermediate host is transparent to the hosts at each end of the connection.

DRCL u/C
P

Brief Description of the Drawing

Fig. 1 is a diagram showing a connection from a process AA on host A to a process CC on host C. Socket routing is utilized to cross the boundary between the networks of type A and type C at host B.

Fig. 2 is a flow diagram showing the operational scenario of Fig. 1 using explicit and implicit routing.

Fig. 3 is a flow diagram showing the modified steps in performing a **connect ()** function to a destination.

Fig. 4 is a flow diagram showing the steps of creating a socket if the host does not have a socket in the specified domain.

Fig. 5 is a flow diagram showing the steps performed at host B.

Fig. 6 is a flow diagram showing the steps of a **connectto ()** function.

Fig. 7 is a more detailed diagram of the socket routing facility of this invention.

DECL u/C
P

Description of the Preferred Embodiment

The following description describes an architecture for routing virtual circuits based on sockets. Although this implies stream sockets, the invention is not limited to stream protocols or to sockets. The

9

concepts of this invention could be applied to similar communication end points that are utilized within other operating systems.

Referring to Fig. 1, a process AA, 10, in a data processing system 11, host A, desires to connect its socket facilities 12 to the process CC, 40, in a data processing system 41, host C. The data processing system 11 is shown as only supporting a particular domain of sockets AF_A, 13, such as TCP, and data processing system 41 is shown as only supporting sockets that exist in the domain having address family C, 43. Since the naming conventions and the underlying transport mechanisms are different between address family A, 13, and address family C, 43, no interconnection can take place without an intermediate facility. The intermediate facility is the socket routing facility 70 in socket layer 32, which exists in data processing system 31, shown as host B.

To describe the initiation of a connection, the process AA, 10, in the data processing system 11, will activate a connection through the sockets programming interface to the general socket code, 12, which in turn goes through the address family specific socket code for AF_A, 13. The necessary data and control information will be handled by the interface and physical access layers, 14. The data will then go out on the network 50 and end up going into data processing system 31, shown as host B, via the interface layer 34, and then through the code for address family A, shown as AF_A, 36.

For comparison, data processing system 21, shown as host D, shows existing internet routing within a single address family, the address family A, AF_A, 23. It should be noted that the cross connection occurs within the address family A, 23. Almost any TCP/IP

implementation can route within its own address family. Likewise, SNA has similar gateway and forwarding capabilities. The cross over as shown in data processing system 21 is independent of the model type of either stream or datagram. It is only dependent upon being within the same network domain.

In data processing system 31, the connection request packets will go through the interface layer HD 13 code 34 to the address family A code, AF_A, 36, through the general socket layer 32, and into the socket routing code 70. The socket routing code facility 70, is where the address mapping and cross connection takes place. The cross connection arrows 37 are shown drawn in the socket routing layer 70 of data processing system 31, as opposed to the cross connection arrows 27 which are shown in the address family code 23 of data processing system 21.

A connection request generated in the socket routing code 70 of data processing system 31 will then HD 13 go down through the address family C code, AF_C, 33, and through the interface layer code 35 for the other network 60, such as SNA. The connection request packets go across the network 60 to the interface HD 13 layer code 44, up to the address family C code, AF_C, 43, continuing through the general socket interface layer code 42 where the connection is registered. Then the process CC, 40, can respond to the connection request in order to establish the connection between cross domain networks.

Figure 7 shows item 70 of Figure 1 in greater detail. Item 701 is the programs and data for controlling the socket routing facility. A connection request to establish socket routing will come in on the sockets for this service, items 704, and 705. The routing agent software, item 703, will accept the

14

connection, which creates a data socket, items 709 - 714. The route request message will come in on that data socket, and the routing agent, 703, will consult its route database, 702, to see if a route is possible. If a route is possible, the routing agent, 703, will consult its route database, 702, on how to establish the route. Then, the routing agent creates a matching data socket (item 710 for item 709, etc.), and connects to the next hop. When the routing agent software receives any replies for further route hops, it forwards them back to the socket routing requestor via the accepted data socket. When all hops are made, the socket routing agent will create a data transfer agent, items 706 - 708, that joins the pairs of data sockets, and forwards data from one to the other and vice versa.

14

The above scenario is further described in the following programming design language code. The following includes examples and uses programs and function names to describe the operational scenario of Fig. 1. The following operational scenario assumes a telnet (or similar program) connected to a remote processing system that is separated by at least one domain boundary. The following uses three machines: "host_A" is connected to "host_B" via TCP, and "host_B" is connected to "host_C" via SNA.

H 13

H 13

```
 */from application view/*
user on host_A says "telnet host_C"
telnet does a gethostbyname for "host_C"
telnet tries to create a socket for domain of "host_C"
     - it fails.
telnet does a getservbyname for sockroute
     - it finds (the only) sockroute available in TCP
     domain
```

T0120X

12

```
telnet invokes sockroute function to get which domain
     to initiate the connection in (or to get a route
     to host_C)
since telnet knows it is now using socket routing it
     uses the (initial domain and routelist) to
     1.  create a socket in its initial domain. (TCP)
     2.  connects to sockaddr of "host_C" telnetd
          -or "connectto routelist telnetd"
when socket connect succeeds, proceed as any
     SOCK_STREAM app would

- alternatively ( with connectto() as "full function")
user on host_A says "telnet host_C"
telnet does a gethostbyname (or getaddrbyname) for
     "host_C" - to see if it exists and to get
     host_C's address
telnet does a "connectto ( host_C:telnetd,
     SOCK_STREAM) - which gets a connected socket.
```

COPYRIGHT IBM CORPORATION 1988

P 14 H 23

The above program design language code is further explained with reference to Figures 2 - 4. The term "telnet" is a remote terminal emulator having the argument "host_C". This invokes the terminal emulator to a remote host, which in this case is "host C", step 201, Fig. 2. "Gethostbyname" is a function call of the telnet program which gets the addressing information for host C, step 203, Fig. 2. The addressing information for host C will include a domain and an address within the domain.

At this point, the routing can be performed either explicitly or implicitly. Explicit action would involve the user code invoking a router function, if the initial attempt to create a socket fails.

13

AT9-88-089

Implicit action would simply be doing a **connectto ()** on the destination address. In explicit routing, the advantage is explicit control by the application. The disadvantages are lack of centralized control, and more complicated user code. In implicit routing, the advantages and disadvantages are just the opposite of those stated above. In implicit routing, the advantages are more centralized control, and less complicated user code. In implicit routing, the disadvantage is that the application does not have direct control.

With explicit routing, Telnet tries to create a socket within that domain, step 204. If the host does not have sockets of that domain, step 205, the socket creation will fail, step 211. At this point, the application, Telnet, invokes a router function, step 213 Fig. 4, if the socket attempt failed, step 211. If the host does have sockets within this domain, the socket attempt will succeed, step 206. If the socket attempt succeeds, the application does a **connect ()**, step 215. The **connect ()** is further shown with reference to Fig. 3. If the **connect ()** succeeds, step 217, Fig. 2, the communication between the two processes proceeds as is typically known in the art, step 207.

If a connect in the same socket domain failed, then (possibly with a socket option set) the socket routing would be invoked. This provides implicit routing, Fig. 3, even in the case of a connection between two domains of the same type, using an intermediate domain of different type.

As shown in Fig. 3. modifying the function **connect ()** enables the **connect ()** to catch those situations in which socket routing is needed to gateway between two like domains using unlike domains.

14

AT9-88-089

If a normal connection, step 301, fails, step 303, and the failure is due to the destination network being unreachable, step 307, then an attempt at implicit routing will be made. This begins with step 311 where a socket route is sought for the destination. If no route is found, then an error is reported, step 315. If a route is found, a connection is made to the socket routing service at the first hop, step 317. Then, a route request is sent, step 319, and the route request replies are received, step 321, until all the hops are connected, step 323. At this time, a connect up request is sent to tell all of the routers to set up the line for data transmission, step 325. After the connect up reply is received, step 327, the peer address of the destination is set for the local socket, step 329, and an indication of success is returned to the invoker of this **connect**, step 331.

Referring back to Fig. 2, a **connectto** function can be added to the generic socket layer code to implement implicit routing from an application level, step 221, Fig. 2. The **connectto** function is called instead of a socket function and a connect function. The function of the socket system call and the function of the connect are combined into the **connectto** function. The advantage of this is that the **connectto** function can handle more addressing issues. Also the **connectto** function does not need to create a socket in the kernel, which may fail, and then have to act upon the failed socket.

The socket parameters of the **connectto** function would include the type and the protocol. Since the previous **connect** call has arguments for the host name, the **connectto** function would take the name of the host in a more portable form, such as the name of the host

15

in a text stream, whereas, **connect** takes the name of the host in a socket structure.

Referring to Fig. 6, the **connectto()** function is further described. If **connectto()** is implemented so that it takes a host name as an argument, then it gets the destination address, step 601. Using this address, the function checks the route table for the destination, step 603. If no route is found, step 605, then an error is returned, step 607. If the destination is in the same domain, and no unlike domains are required for gateways, step 609, then a socket is created in the same domain, step 611. A normal connection is established to the destination, step, 612. The route for communication is then established, step 613.

If the destination is not the same domain or unlike domains are required for gateways, step 609, then a socket is created in the domain of the first hop, step 615. A connection to the socket routing service at the first hop is then established, step 617. A route request is sent, step 619, and a reply to the request is received, step 621, until all hops are connected, step 623. After this, a connect up request is sent, step 625, and its reply is received, step 627. The peername of the destination is set for the local socket, step 629. The route is now available for normal communications, step 613.

With the following modifications, referred to as socket routing, the creation of a socket can continue, step 213, as shown in Fig. 4, when the host does not have a socket in the specified domain, step 205, Fig. 2. The modifications take place at the client side, host_A. Host_C is referred to as the server.

H 13

The telnet application performs a "getservbyname" function for the socket routing service, step 401,

16

AT9-88-089

Fig. 4. If, for example, the host only has sockets in the TCP domain, telnet will find the only socket route available in the TCP domain, step 403. Next, telnet uses the sockroute function, step 405, to determine the route and what domain of socket to create, step 406. Then, the socket is created for the initial hop of the route, step 412, and then the connection would be set up, step 413. At this point, the application can talk to the host as it otherwise would have with any other socket stream, and in this case, using the telnet data stream, step 414.

H 13

H 13 3

H 13 3

Assuming the route initialization is done by a daemon or library function on host_A (and not kernel code), then host_A's socket code doesn't really have much to do with socket routing. Basically, if socket routing is performed outside of the operating system kernel on host_A, then no changes to host_A's socket code need to be made.

P

H 13

The following programming design language code, and the following description with reference to Figure 5 describes what happens on host_B.

TO170X

```
 */ on host_B /*
sockroute daemon receive connection from host_A
     (asking for connection to host_C)
sockroute daemon consults route table -or route list
     provided with connection request.
sockroute daemon decides to connectto to host_C via
SNA socket
     (since it is last hop, it doesn't need to connect
     to a sockroute daemon on host_C)
when connection completes, host_B sockroute daemon
     1.  sends response back to socket routing on
         host_A
     2.  cross connects the TCP and SNA sockets on
         host_B
```

17

AT9-88-089

when routing on host_A receives response, it pulls out of the way, leaving telnet connected all the way to host_C

COPYRIGHT IBM CORPORATION 1988

P

Essentially, the above code describes the scenario in which a service waits around for a connection. With reference to Fig. 5, the sockroute daemon, which runs on host_B, receives connections from other processes requesting its services, step 501. The sockroute daemon is analogous to a telephone operator who is requested to make a connection to another person from a caller. The requesting process, caller, supplies the sockroute daemon, operator, with the necessary connection information in order to make the connection, step 503. Once the sockroute daemon makes the connection, the sockroute daemon leaves the connection. If this connection leads to the final destination, step 505, no other sockroute daemons on a next host need to be called, and the sockroute daemon connects to the final host destination via a SNA socket, step 507. However, it is possible to have multiple sockroute daemons, operators, that are needed to make a connection from a first host to a final host destination. If this connection does not lead to the final host connection, then another sockroute daemon on a next host must be called, step 506, and the above steps repeated.

H 13

H 13

The sockroute daemon on host_B then sends a response back to the socket routing service on the originating host, host_A, step 509. Host_B cross connects the TCP and SNA sockets on host_B, step 511. When the routing service on host_A receives the response, host_B pulls out of the way. This leaves a

18

H 13 telnet connection all the way from host_A to host_B, step 513.

H 13 It should be noted that since host_C is the end of the line, its socket layer is entirely unaffected for data transfer purposes.

There is a function called getpeername () that is part of the sockets programming interface. A socket can also be queried as to which service is connected H 13 to it. For example, if host_C queried its socket to determine which service at the other end it was connected to, the response would be the intermediate H 13 host, host_B, instead of the actual service at the other end of the connection which in this example is H 13 host_A. Therefore, the getpeername would need input from the socket routing code at both ends of the connection, as well as some kernel changes, for it to work in a transparent fashion. For transparency, the H 13 getpeername would respond with host_A, the real end of L L the completed connection, if the socket in host_C was queried as to the party at the other end of the connection.

The details of the address mapping and socket routing facilities within the socket layer 32, which effectuates the cross domain connections, are described hereafter.

Gatewaying of socket based protocols is achieved by looping two sockets together at the top end. Such a mechanism would allow a router to create a path that would cross domain boundaries. A router in this context would be program code that would decide how to get to one data processing system to the other such as in the internet layer of TCP/IP. SNA also has similar code. The mechanism for looping two sockets together at the top end would not require file descriptors, or

process switching time on the connecting node, once the connection is established.

The following illustrates the changes to the socket layer interface of an operating system, such as the AIX operating system that utilizes the Berkeley sockets, that may be made to implement socket routing of this invention. These changes include the following:

* modify "connect" to catch cross domain connects
* add "connectto" to implement implicit routing from application level.
* as an option, create library functions for routing
* modify socket buffer handling, etc. to allow cross connections without process intervention
* as an option, add function so getpeername works transparently
* define socket routing protocol and messages (in kernel or as a daemon)
* if needed, modify nameserver for domain gateways and routing info.

If **connectto** is not used to hide the routing from the user in a library, it is also possible to create library functions to perform the routing. However, the user will require a facility to figure out which machine has a socket routing daemon to service an intermediary. These functions(s) would allow a user program to invoke socket routing with minimal effort. Possible function to be defined are:

* "get_route" - user program asks for route (useable by connect())
* "get_type_of_socket_I should_open_to_get_to_host" - done against the return from "get_route" -or does implicit "get_route".

P0 13B B

* connectto" - (1) looks up route, (2) creates a socket in proper domain, (3) established connection.

P1 P2

1113B

P1 P2 1113B

```
i.e., instead of
  hp = gethostbyname(host);
  (fill in sockaddr from hp ...)
  so = socket(AF_XX, SOCK_STREAM,0);
  connect(so, sockaddr, sockaddrlen);
a program does
  so = connectto( host, SOCK_STREAM, 0);
```

P

In addition, modifying socket buffer handling will allow cross domain connections without process intervention. Previously, a socket is set up such that when data arrives, the data is stored in a queue while the data waits for a process to read it. At the gateway, the socket routing machine, when data arrives from one end of the connection, the data has to be automatically sent out the other side to the other end of the connection, and vice versa.

A current implementation of socket buffering would require that a process be running against all the sockets that are cross connected. A more efficient means would be to add this cross connection at a socket buffer layer, so that no process scheduling needs to be done to send the data on its way. In either case, flags are added to the socket data structures.

As previously mentioned, additional function is added to the "getpeername" function to enable the intermediate host to appear transparently in the connection between the originating host destination and the final host destination. Previously, the socket peer address has been handled by protocol dependent means. A change is required so that

21

getpeername() works correctly. The change involves having the peer address propagated by the route daemons, in both directions. Then the routing code at each end of the connection would do a "set peer address" operation, which would override the protocol's peer address function.

The socket routing facility of this invention also requires a socket routing protocol and messages. It is desirable that the socket routing code handle routing in a flexible manner. To achieve this, a preferred embodiment of this invention has a socket routing daemon on each machine that is an interdomain gateway. The daemon would be listening on well-known socket(s) for routing requests. When a request came in (via a connecting socket) the routing daemon would examine the request and perform the desired action.

P These requests (and their responses) are as follows:

Po
P1 Messages For Socket Routing Protocol

- and the information that goes with each message

Po 13
P1
route request - sent to request a route be set up

- originator address
- hop destination address
- flag for intermediate or final hop

Po 13
P1
route request reply - received to indicate completion and success/fail of route request

- status for success or failure

Po 13
P1 21 22
connectup request - sent to establish normal data pathway

- <none>

22

Po 13
P1 9

<u>connectup reply</u> - received to indicate completion and success/fail of connectup request
- status for success or failure

P

The socket routing service code is used to perform routing at the intermediate nodes, i.e. the gateway node. When a request for service arrives at the gateway machine, such as for any other socket connection, the request for service would arrive at a particular socket which would be the socket of the socket routing daemon. The process with this particular socket open could be either in the kernel or running as a user level process.

Therefore, the socket routing service code can be created as a daemon or in the kernel. Preferably, the socket routing service code will exist mostly or completely as a daemon. Some minor parts, such as ioctls (input output controls) to tie sockets together, may exist as part of the kernel. However, these minor parts support the daemon, and are not really a part of the socket routing service code. As an alternative, it is also possible to put the routing implementation part (as opposed to the route figuring out part) in the kernel, which would save process context switch time.

Another modification may be made to implement the socket routing of this invention. The nameserver may be modified for domain gateways and routing information. The (name) domain name server needs to have a type of data for inter(socket) domain gateways. It may also be desirable for it to find gateways when looking up a host address. It would be desirable if it would flag the fact that a host requires an inter(socket) domain gateway to get to it.

23

While the invention has been particularly shown and described with reference to a preferred embodiment including sockets, the underlying idea of cross domain connections could be achieved with other operating systems having other communication endpoints other than sockets. It will be understood by those skilled in the art that various changes in form and detail may be made without departing from the spirit and scope of the invention.

CM I claim: Claims

24

AT9-88-089

Claims

Claim 1. A system for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, said system comprising:

at least one communication end point object in a layer of said first data processing system and in said layer of said second data processing system;

means, independently of an application running on either of said data processing systems, for automatically routing, in said layer having said at least one communication end point object, a connection between said first processing system and said second processing system; and

means for communicating over said routed connection between said first data processing system and said second data processing.

Claim 2. The system of claim 1 wherein the first network domain is a Transmission Control Protocol and the second network domain is a Systems Network Architecture.

Claim 3. A system for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, said system comprising:

at least one communication end point object in a ~~layer of~~ said first data processing system, in

AT9-88-089

AT

said layer of an intermediate data processing system, and in said layer of said second data processing system;

means, in said intermediate data processing system, for automatically routing, in said layer having said at least one communication end point object, a connection between said first processing system and said second processing system; and

means for communicating through said routed connection in said intermediate processing system between said first data processing system and said second data processing system.

Claim 4. The system of claim 3 wherein said means for communication in said intermediate processing system immediately sends any data received from one end of said routed connection to said other end of said routed connection.

Claim 5. The system of claim 3 wherein said at least one communication end point object is a socket in a socket layer of each of said data processing systems.

Claim 6. A system for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, said system comprising:

at least one socket in a socket layer of said first data processing system and in said layer of said second data processing system;

means, independently of an application running on either of said data processing systems, for automatically routing, in said socket layer, a connection between said first data processing system and said second data processing system; and

means for communicating through said socket connection between said first data processing system and said second data processing system.

Sub B2

Claim 7. A method for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, said method comprising:

creating, by said first data processing system, a socket in said second network domain; and

invoking a routing facility to automatically connect a socket in said first data processing system to said created socket in said second data processing system when said socket is created in said second network domain; and

communicating over said socket connection ~~between~~ said socket in said first data processing system in said first domain and said created socket in said second data processing system in said second domain.

Claim 8. A method for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, said method comprising:

determining a means to make a connection between a first socket in said first data processing system and said second data processing system;

creating a second socket in the domain of the second data processing system to establish the determined connection; and

communicating over said determined connection between said socket in said first data processing system in said first domain and said created socket in said second domain of said second data processing system.

Claim 9. An operating system for use with a plurality of data processing systems for communicating between a first data processing system in a first network domain and a second data processing system in a second network domain, said operating system comprising:

at least one socket in a socket layer of said first data processing system and in said layer of said second data processing system;

means, independently of an application running on either of said data processing systems, for automatically routing, in said socket layer, a connection between said first data processing system and said second data processing system; and

means for communicating through said socket connection between said first data processing system and said second data processing system.

Print Of Drawing
As Original Filed

1/7
G. L. Owens
MDS AT9-88-089

mG162117956

07 304696

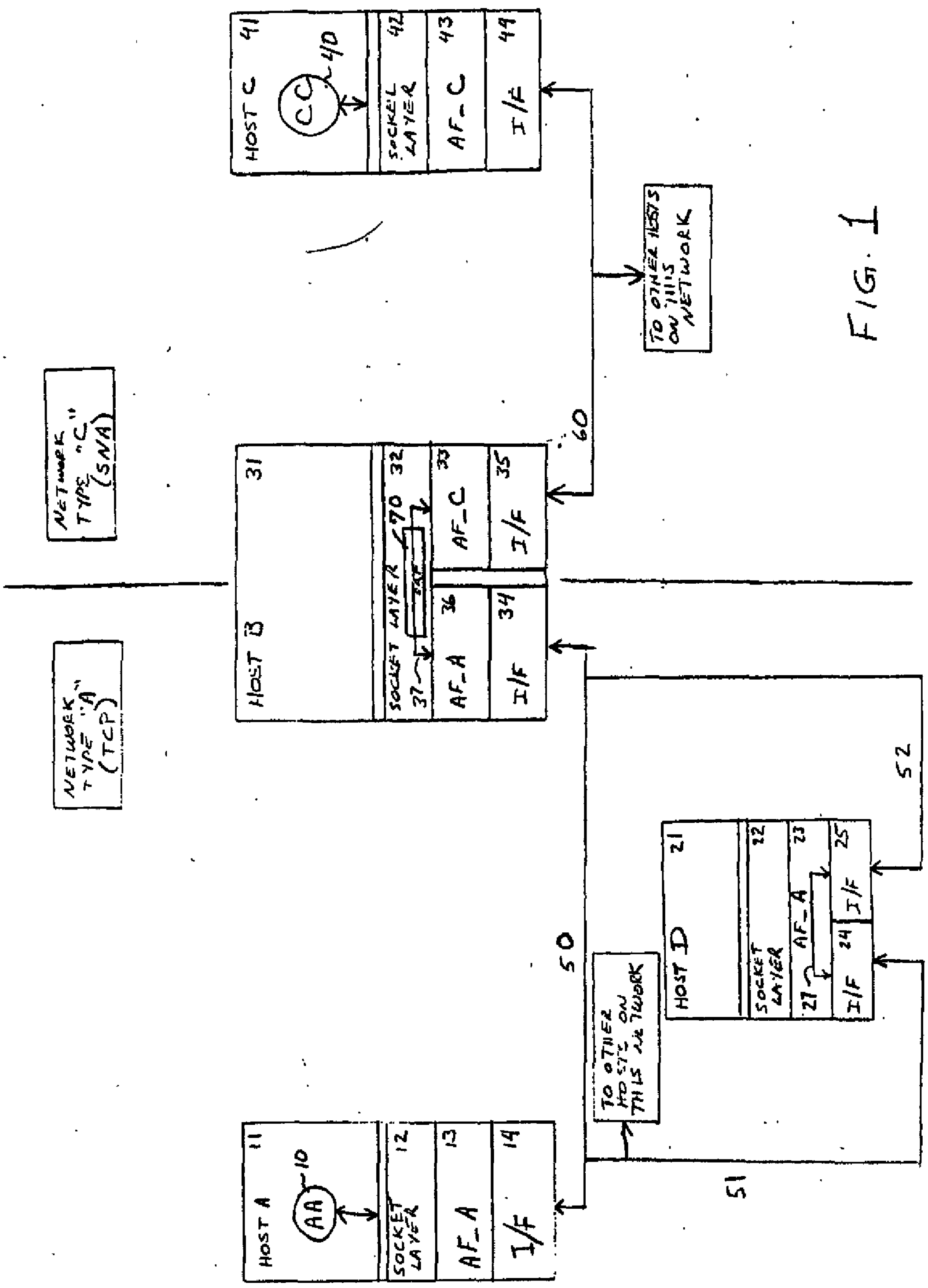


FIG. 1

Print Of Drawing
As Original Filed

07 304696

2/7
AT9-88-089

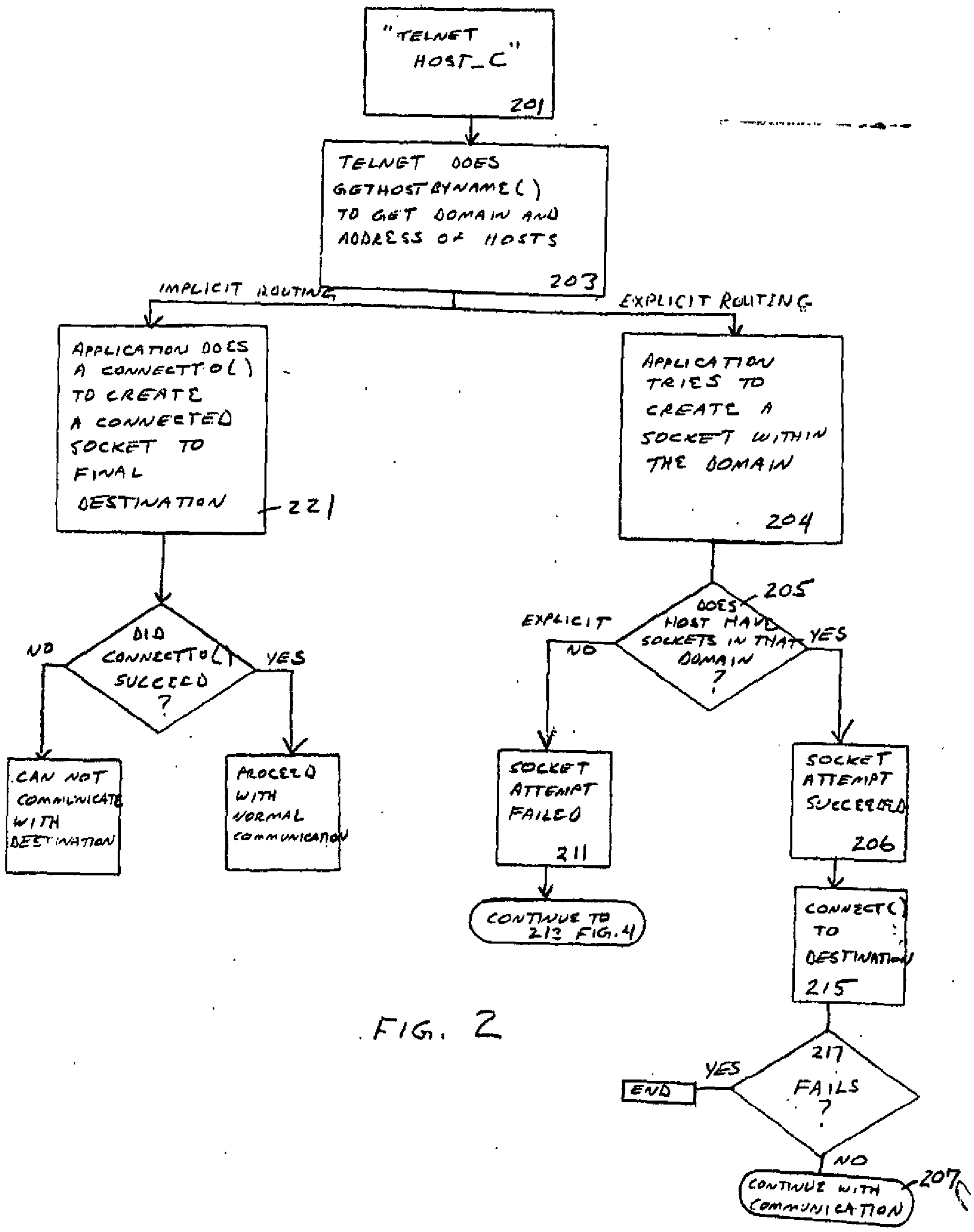


FIG. 2

Print Of Drawing
As Original Filed

07 304696

3/7
A19-88-089

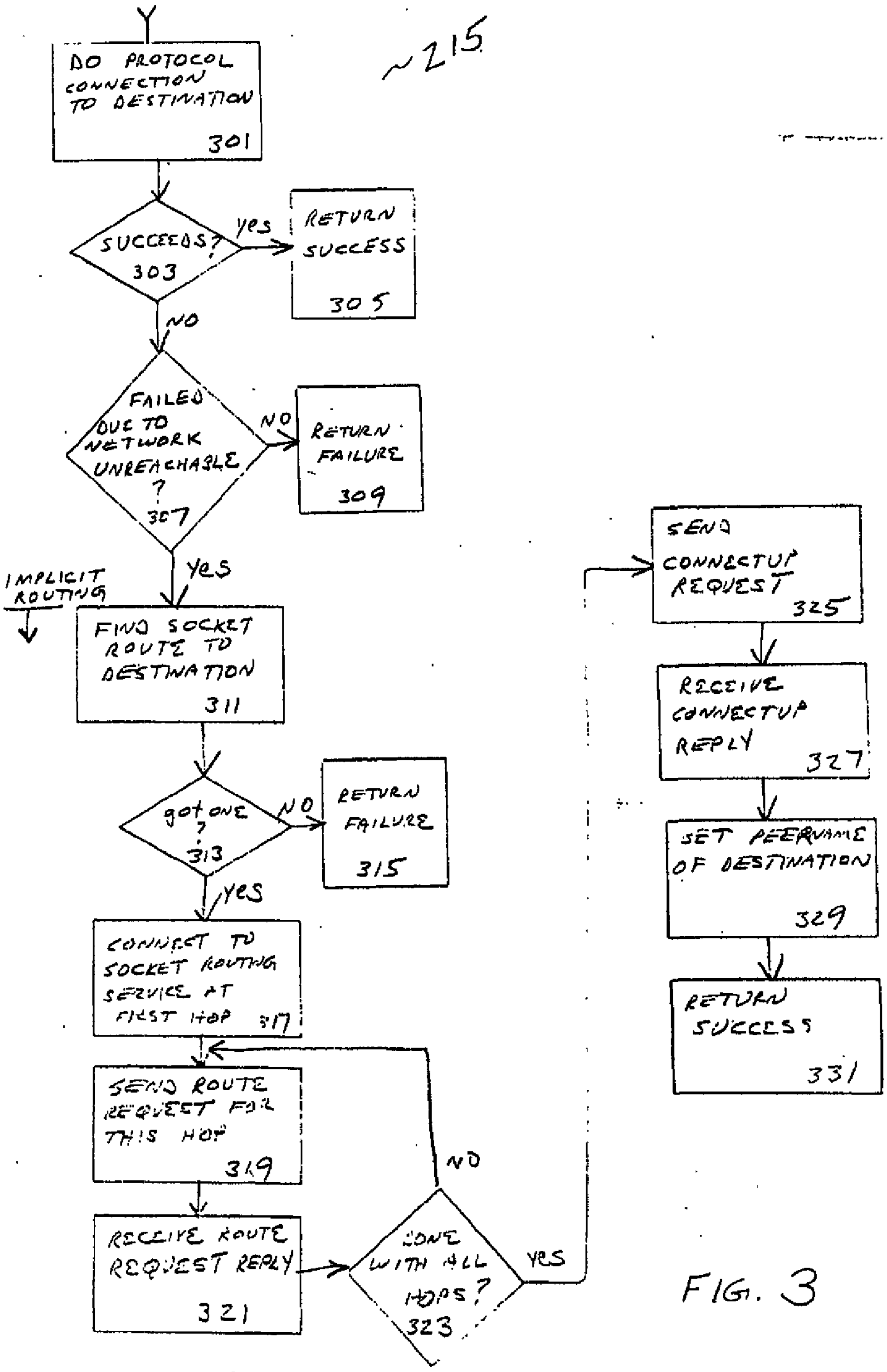


FIG. 3

Print Of Drawing
As Original Filed

4/7
AT9-88-089

07 304696

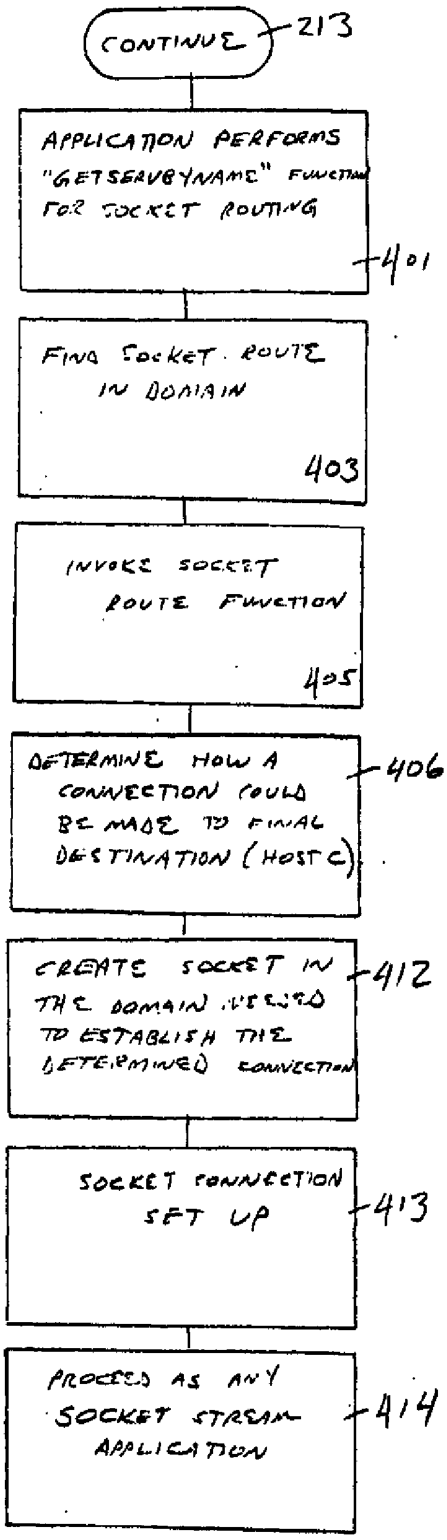


FIG. 4

Print Of Drawing
As Original Filed

5/7
AT9-88-089

07 304696

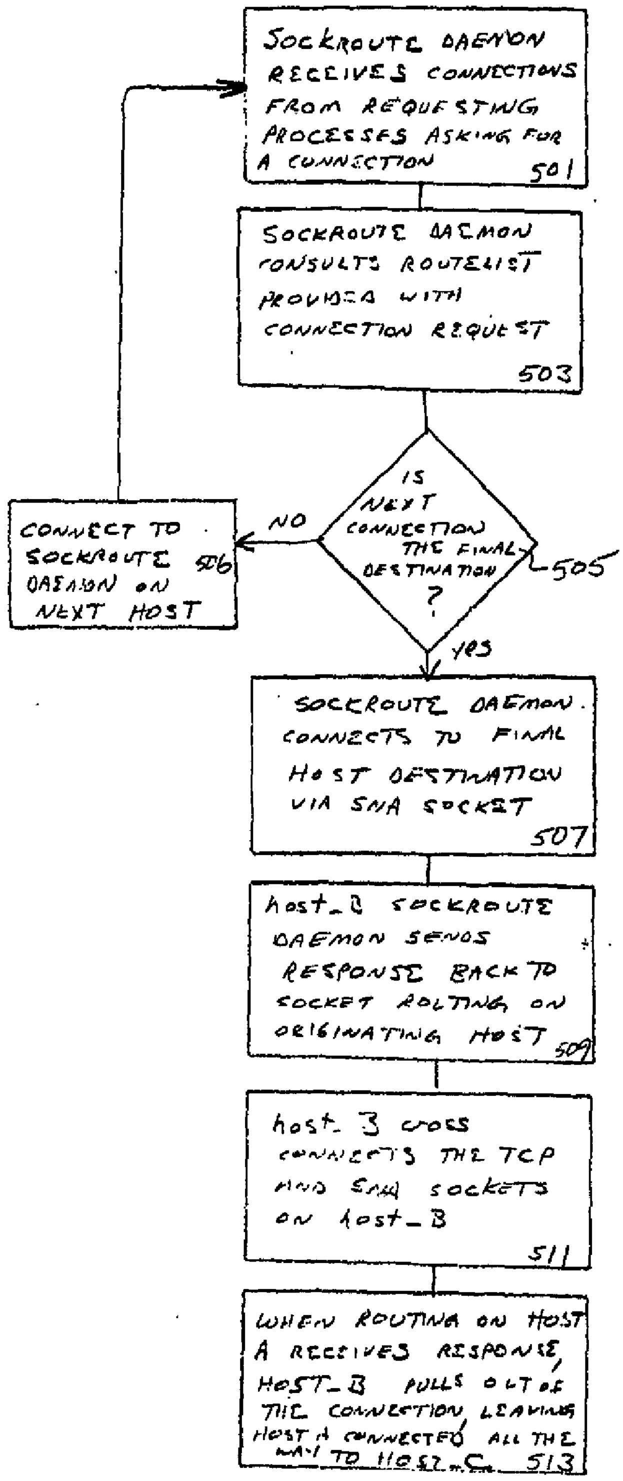


FIG. 5

Print Of Drawing
As Original Filed

07 304696

6/7
AT9-88-089

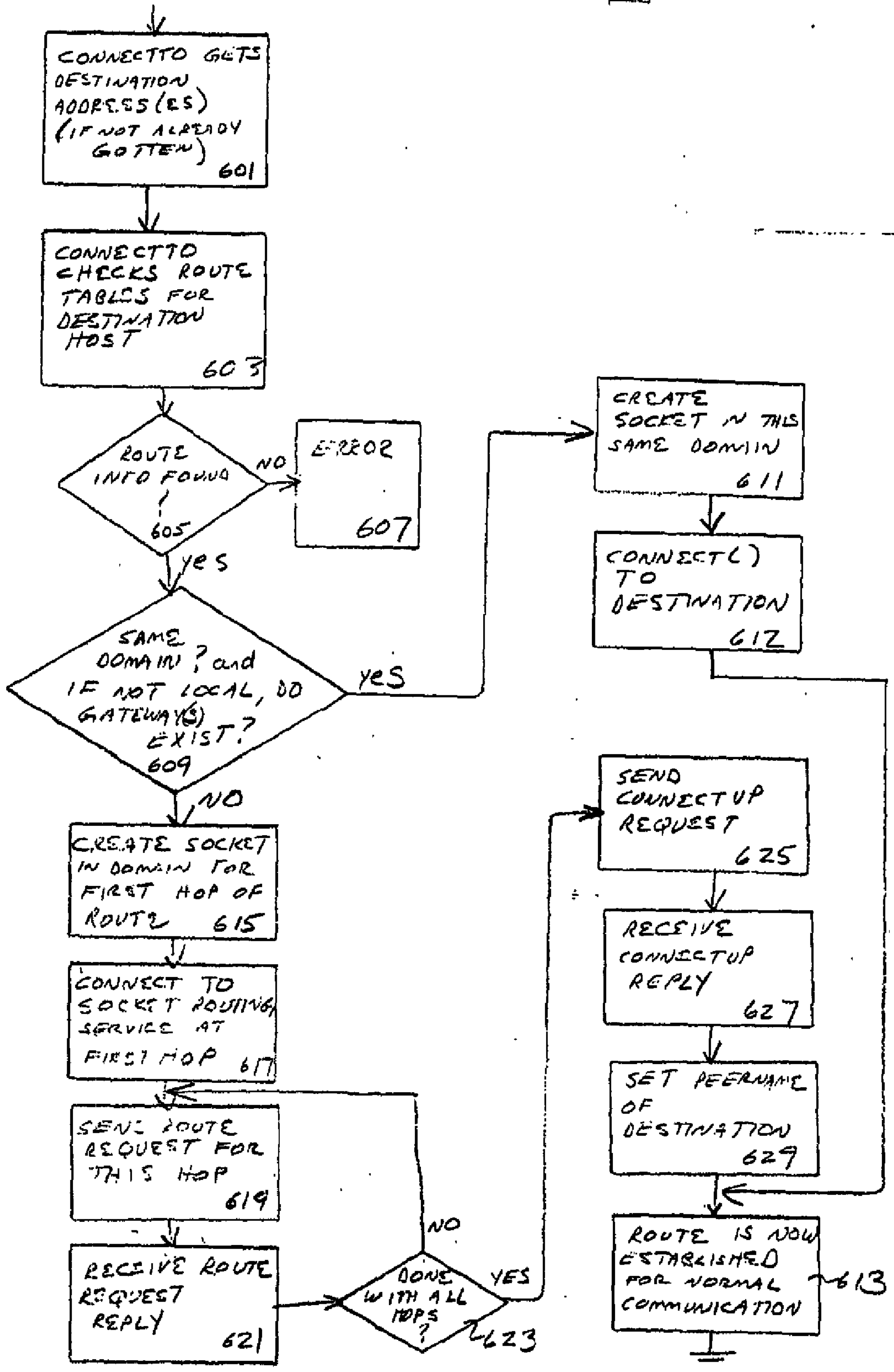


FIG. 6

Print Of Drawing
As Original Filed

7/7
AT9-88-089

304696

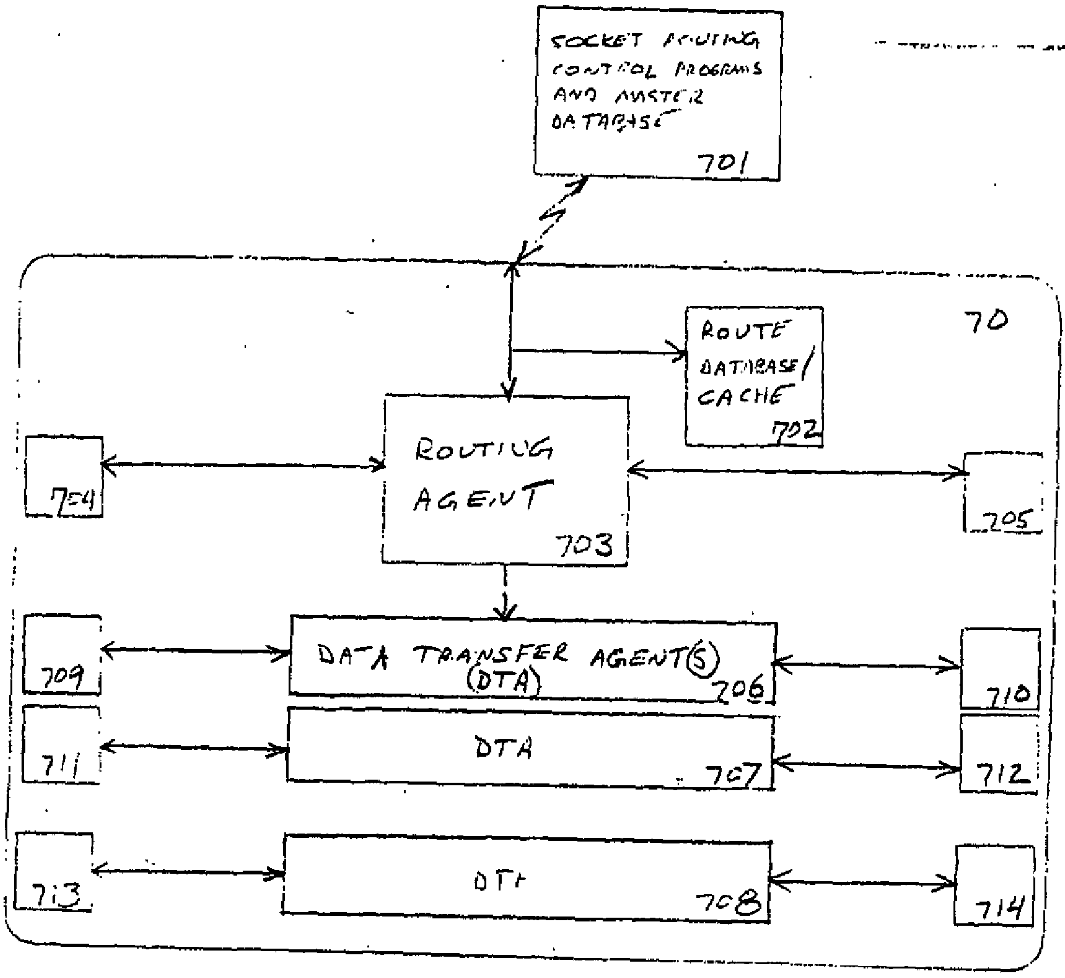


FIG. 7